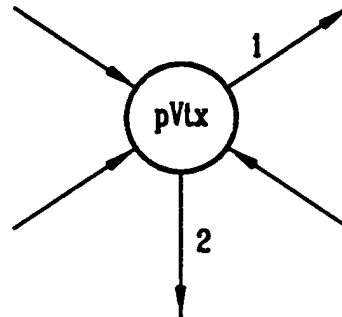


# VtxVisitOutEdgeClient

---

## Diagram

Walk outgoing Edges



# VtxVisitOutEdgeClient

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxVisitOutEdgeClient( pVtx, offset, pBlk )
PVTX      pVtx;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The `VtxVisitOutEdgeClient` function walks all the outgoing edges linked to the *Vertex* `pVtx` and calls an *Edge* client function for each edge visited. The *Block* `pBlk` contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

<code>pVtx</code>	-	Pointer to a structure of type <i>Vertex</i> .
<code>offset</code>	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

## See Also

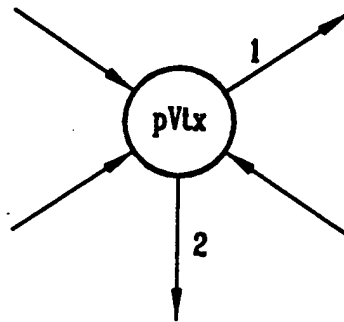
`VtxVisitEdge`, `VtxVisitEdgeClient`, `VtxVisitInEdge`, `VtxVisitInEdgeClient`, `VtxVisitOutEdge`

# VtxVisitOutEdge

---

## Diagram

Walk outgoing Edges



# VtxVisitOutEdge

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxVisitOutEdge( pVtx, pBlk )
PVTX      pVtx;
PBLK      pBlk;
```

Friend Function

## Purpose

The VtxVisitOutEdge function walks all the outgoing edges linked to the *Vertex* pVtx and calls an *Edge* function for each edge visited. The *Block* pBlk contains the function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The edge function may return a value but it is ignored.

## See Also

VtxVisitEdge, VtxVisitEdgeClient, VtxVisitInEdge, VtxVisitInEdgeClient  
VtxVisitOutEdgeClient

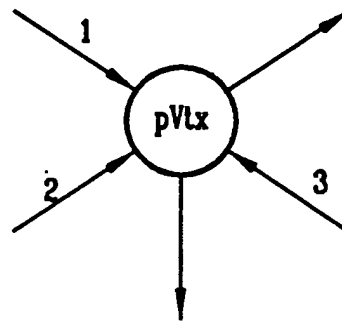


## VtxVisitInEdgeClient

---

### Diagram

Walk Incoming Edges



# VtxVisitInEdgeClient

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void          VtxVisitInEdgeClient( pVtx, offset, pBlk )
PVTX          pVtx;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The `VtxVisitInEdgeClient` function walks all the incoming edges linked to the *Vertex* `pVtx` and calls an *Edge* client function for each edge visted. The *Block* `pBlk` contains the client function and an optional list of arguments.

## Parameter - Description

---

<code>pVtx</code>	-	Pointer to a structure of type <i>Vertex</i> .
<code>offset</code>	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

## See Also

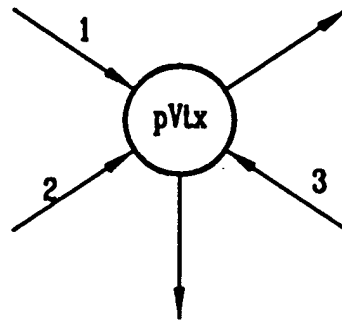
`VtxVisitEdge`, `VtxVisitEdgeClient`, `VtxVisitInEdge`, `VtxVisitOutEdge`, `VtxVisitOutEdgeClient`

## VtxVisitInEdge

---

### Diagram

Walk Incoming Edges



## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void          VtxVisitInEdge( pVtx, pBlk )
PVTX          pVtx;
PBLK          pBlk;
```

Friend Function

## Purpose

The *VtxVisitInEdge* function walks all the incoming edges linked to the *Vertex* *pVtx* and calls an *Edge* function for each edge visited. The *Block* *pBlk* contains the function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

<i>pVtx</i>	-	Pointer to a structure of type <i>Vertex</i> .
<i>pBlk</i>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The edge function may return a value but it is ignored.

## See Also

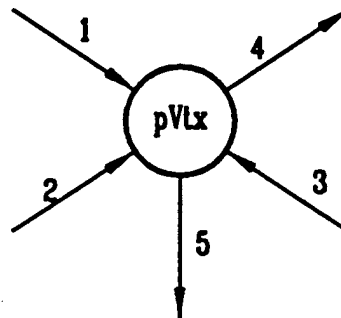
*VtxVisitEdge*, *VtxVisitEdgeClient*, *VtxVisitInEdgeClient*, *VtxVisitOutEdge*, *VtxVisitOutEdgeClient*

# VtxVisitEdgeClient

---

## Diagram

Walk all Edges



# VtxVisitEdgeClient

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void          VtxVisitEdgeClient( pVtx, offset, pBlk )
PVTX          pVtx;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The VtxVisitEdgeClient function walks all the client edges linked to the *Vertex* pVtx and calls an *Edge* client function for each edge visited. The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
offset	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

## See Also

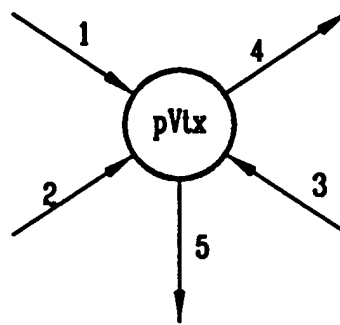
VtxVisitEdge, VtxVisitInEdge, VtxVisitInEdgeClient, VtxVisitOutEdge, VtxVisitOutEdgeClient

# VtxVisitEdge

---

## Diagram

Walk all Edges



# VtxVisitEdge

---

## Summary

```
#include "coljects.h"
#include "vtxmac.h"
```

```
Void      VtxVisitEdge( pVtx, pBlk )
PVTX      pVtx;
PBLK      pBlk;
```

Friend Function

## Purpose

The VtxVisitEdge function walks all the edges linked to the *Vertex* pVtx and calls an *Edge* function for each edge visited. The *Block* pBlk contains the function and an optional list of arguments.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The edge function may return a value but it is ignored.

## See Also

VtxVisitEdgeClient, VtxVisitInEdge, VtxVisitInEdgeClient, VtxVisitOutEdge, VtxVisitOutEdgeClient



# VtxStackSetup

---

## Summary

```
#include "cobjects.h"  
#include "vtxmac.h"
```

```
Void          VtxStackSetup( pVtx, inStack, outStack, stackVtx, i )  
PVTX         pVtx;  
MediumInt    *inStack;  
MediumInt    *outStack;  
PVTX         *stackVtx;  
MediumInt    *i;
```

Friend Function

A macro is available for this function

## Purpose

The VtxStackSetup function is a private function used the *Graph* class.

## Return Value

No return value

## Notes

This is a function only to be used by the *Graph* class and is used by the topological sort function.

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxStackSetup function.

# VtxSendDestroy

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxSendDestroy( pVtx )
PVTX      pVtx;
```

Public Function

## Purpose

The VtxSendDestroy function sends a message to the client of the *Vertex* pVtx asking it to destroy the vertex. The *Vertex* client function will receive this message and should deinitialize or destroy the vertex. This message function should be included in the *Vertex* client message array.

Parameter	-	Description
-----------	---	-------------

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

No return value

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxSendDestroy function.

# VtxInGrf

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Bool      VtxInGrf( pVtx )
PVTX      pVtx;
```

Public Function

A macro is available for this function

## Purpose

The VtxInGrf function determines if the *Vertex* pVtx is linked to a *Graph*.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

The return value from the VtxInGrf function is True if the *Vertex* pVtx is linked to a *Graph*, otherwise False if returned.

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxInGrf function.

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxInit( pVtx )
PVTX      pVtx;
```

Public Function

## Purpose

The VtxInit function initializes the *Vertex* object. The VtxInit function should be the first function called when using the *Vertex* class.

Parameter	-	Description
-----------	---	-------------

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

No return value

## Notes

The last function to call when using the *Vertex* class is VtxDeInit.

## See Also

VtxDeInit, VtxDestroy

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxInit function.

# VtxGetGrf

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
PGRF      VtxGetGrf( pVtx )
PVTX      pVtx;
```

Friend Function

A macro is available for this function

## Purpose

The VtxGetGrf function returns the a pointer to the graph that the *Vertex* pVtx is linked to.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

The return value from the VtxGetGrf function is a pointer to the *Graph* structure contained by the *Vertex* pVtx or NULL if not linked to a graph.

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxGetGrf function.

# VtxGetFirstOut

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
PEDG      VtxGetFirstOut( pVtx )
PVTX      pVtx;
```

Public Function

A macro is available for this function

## Purpose

The VtxGetFirstOut function returns the first outgoing *Edge* linked to the *Vertex* pVtx or NULL if none exist.

Parameter	-	Description
-----------	---	-------------

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

The return value from the VtxGetFirstOut function is a pointer to an *Edge* structure or NULL. This edge is the first in a list of outgoing edges linked to the *Vertex* pVtx.

## See Also

VtxGetFirstIn

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxGetFirstOut function.

# VtxGetFirstIn

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
PEDG      VtxGetFirstIn( pVtx )
PVTX      pVtx;
```

Public Function

A macro is available for this function

## Purpose

The VtxGetFirstIn function returns the first incoming *Edge* linked to the *Vertex* pVtx or NULL if none exist.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

The return value from the VtxGetFirstIn function is a pointer to an *Edge* structure or NULL. This edge is the first in a list of incoming edges linked to the *Vertex* pVtx.

## See Also

VtxGetFirstOut

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxGetFirstIn function.

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
POBJ      VtxGetClient( pVtx, offset )
PVTX      pVtx;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The VtxGetClient function returns the client of the *Vertex* pVtx.

Parameter	-	Description
-----------	---	-------------

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
offset	-	The distance in bytes between the <i>Vertex</i> pVtx and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the VtxGetClient function is a pointer to the client of the *Vertex* pVtx.

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxGetClient function.



# VtxFindOutEdgClient

## Notes

The client function should return True for the VtxFindOutEdgClient function to return True.

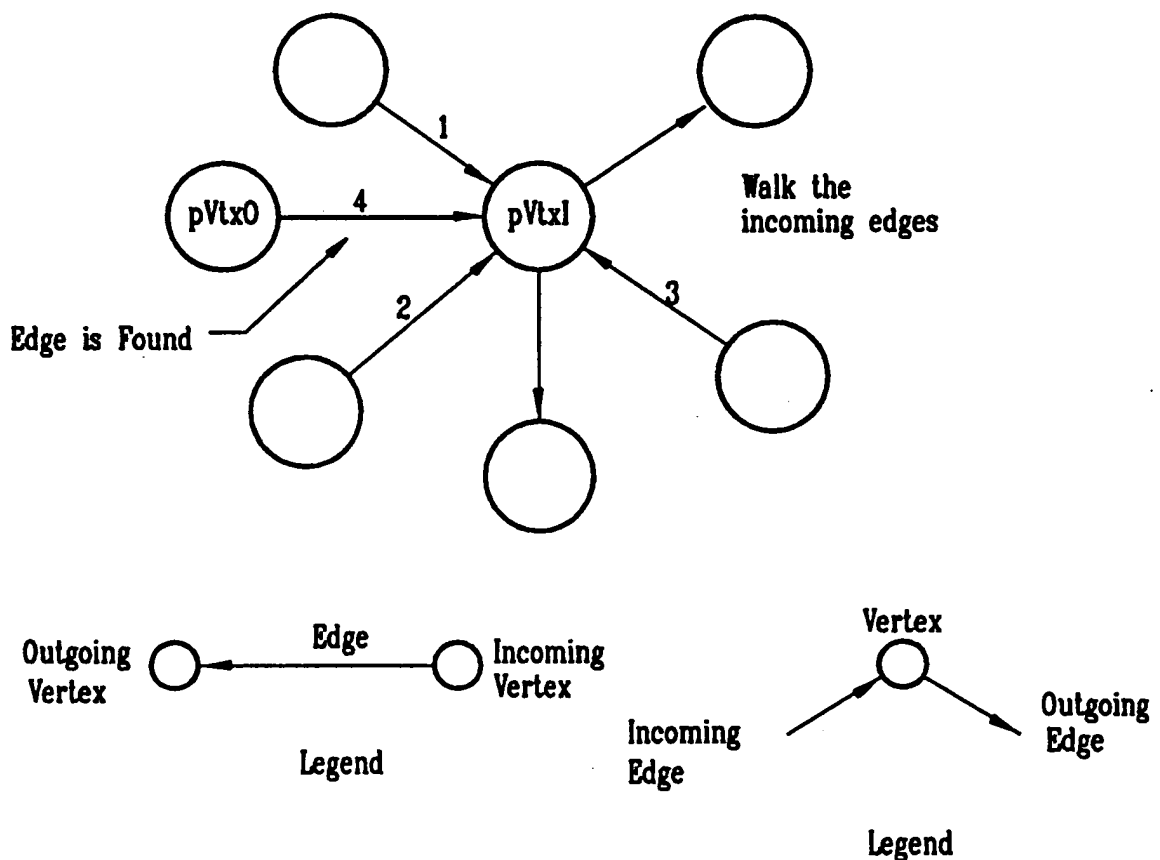
The optional function will only be called if the edge links to the vertices pVtxI and pVtxO.

## See Also

VtxFindOutEdg, EdgCompareInVtx

## Diagram

Is pVtx0 linked to my incoming Edge?



# VtxFindOutEdgClient

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
POBJ      VtxFindOutEdgClient( pVtxO, pVtxI, offset, pBlk )
PVTX      pVtxO;
PVTX      pVtxI;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The *VtxFindOutEdgeClient* function walks all the incoming edges linked to the *Vertex* *pVtxI* and returns the client of the edge that has the *Vertex* *pVtxO*, as an outgoing edge. If no edge exists NULL is returned.

If the caller requires that the comparison should have additional criteria to linkage, an *Edge* client function can be given to call as the incoming edges are visited. If no function call is required a function NULL should be given.

The *Block* *pBlk* contains the client function and an optional list of arguments. The function should return a boolean (True/False) value.

## Parameter - Description

---

<i>pVtxO</i>	-	Pointer to a structure of type <i>Vertex</i> where this vertex has the search edge as an outgoing edge.
<i>pVtxI</i>	-	Pointer to a structure of type <i>Vertex</i> where this vertex has the search edge as an incoming edge.
<i>offset</i>	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
<i>pBlk</i>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

*offset*: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the *VtxFindOutEdgClient* function is a pointer to the *Edge* client structure for the edge that links to the vertices *pVtxI* and *pVtxO*. If no edge is found then NULL is returned.

# VtxFindOutEdg

## Notes

The function should return True for the VtxFindOutEdg function to return True.

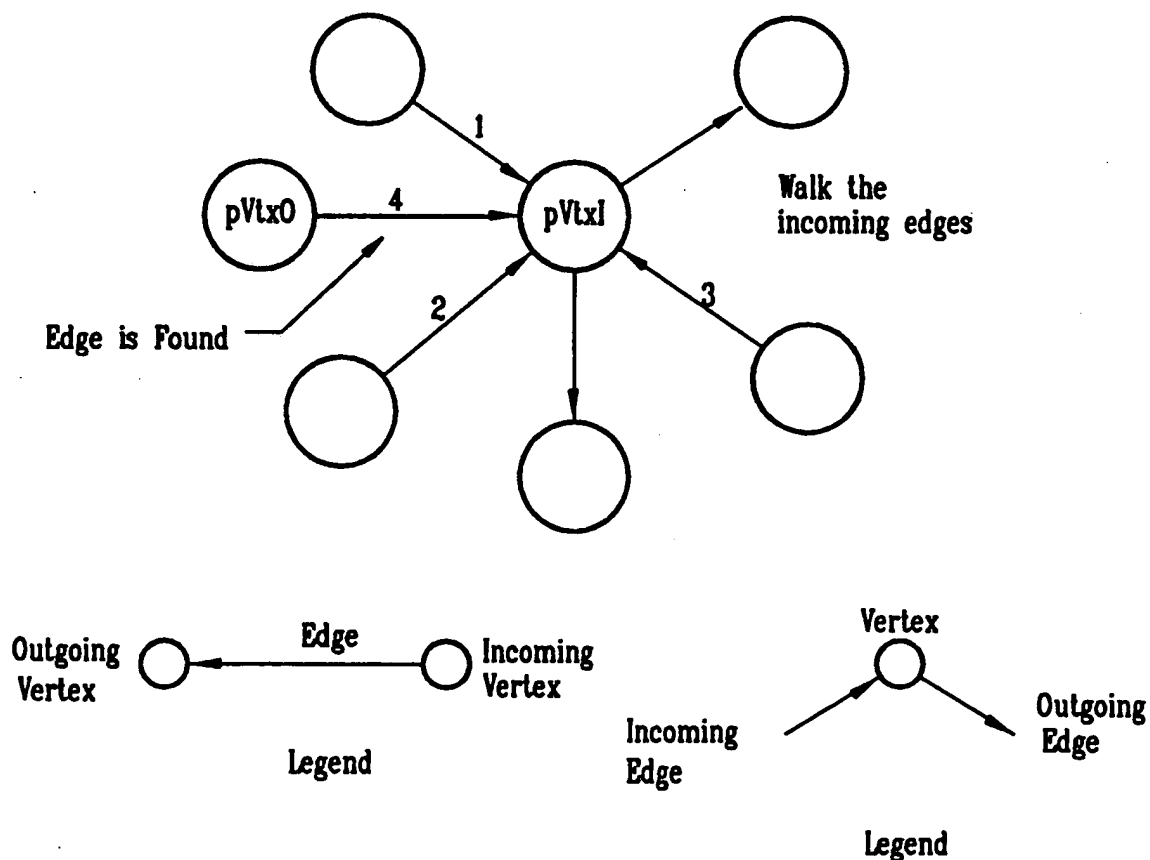
The optional function will only be called if the edge links to the vertices pVtxI and pVtxO.

## See Also

VtxFindOutEdgClient, EdgCompareInVtx

## Diagram

Is pVtx0 linked to my incoming Edge?



# VtxFindOutEdg

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
PEDG      VtxFindOutEdg( pVtxO, pVtxI, offset, pBlk )
PVTX      pVtxO;
PVTX      pVtxI;
MediumInt offset;
PBLK      pBlk;
```

Private Function

## Purpose

The VtxFindOutEdge function walks all the incoming edges linked to the *Vertex* pVtxI and returns the edge that has the *Vertex* pVtxO, as an outgoing edge. If no edge exists NULL is returned.

If the caller requires that the comparison should have additional criteria to linkage, an *Edge* client function can be given to call as the incoming edges are visited. If no function call is required a function NULL should be given.

The *Block* pBlk contains the client function and an optional list of arguments. The function should return a boolean (True/False) value.

## Parameter - Description

---

pVtxO	-	Pointer to a structure of type <i>Vertex</i> where this vertex has the search edge as an outgoing edge.
pVtxI	-	Pointer to a structure of type <i>Vertex</i> where this vertex has the search edge as an incoming edge.
offset	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

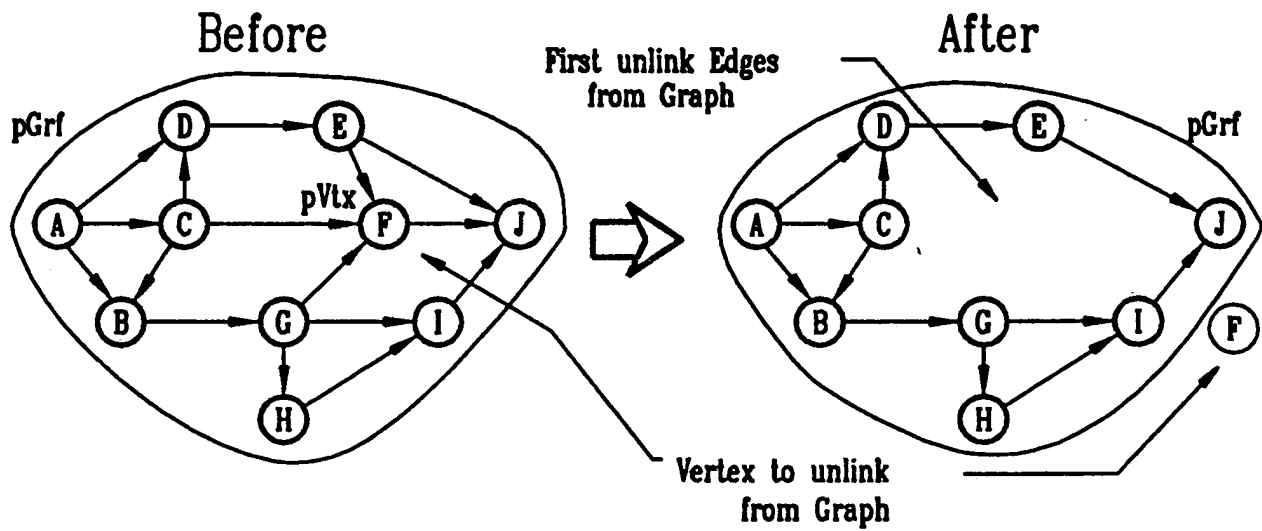
offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the VtxFindOutEdg function is a pointer to an *Edge* structure. The edge returned is linked to the vertices pVtxI and pVtxO. If no edge is found then NULL is returned.

# VtxDisconnectFromGrf

## Diagram



# VtxDisconnectFromGrf

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void          VtxDisconnectFromGrf( pVtx )
PVTX          pVtx;
```

Public Function

A macro is available for this function

## Purpose

The VtxDisconnectFromGrf function unlinks the *Vertex* pVtx from a *Graph*. The vertex must not have any linkage to edges.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

No return value

## Notes

[pVtx must be linked to a graph.]

[pVtx must have no edges.]

## See Also

VtxConnectToGrf

# VtxDestroy

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxDestroy( pVtx )
PVTX      pVtx;
```

Public Function

## Purpose

The VtxDestroy function deallocates the memory used by the object and deinitializes the *Vertex* object. The *Vertex* pVtx should not be referenced after this function call since its memory will have been deallocated.

Any vertices linked to pVtx will be unlinked and pVtx will also be unlinked from its graph.

## Parameter - Description

---

pVtx - Pointer to a structure of type *Vertex*.

## Return Value

No return value

## Notes

[pVtx must not have a sub-object.]

## See Also

VtxDeInit, VtxInit

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxDestroy function.

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxDeInit( pVtx )
PVTX      pVtx;
```

Public Function

## Purpose

The *VtxDeInit* function deinitializes the *Vertex* object. The *VtxDeInit* function should be the last function called when using the *Vertex* class.

Parameter	-	Description
-----------	---	-------------

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

No return value

## Notes

The first function to call when using the *Vertex* class is *VtxInit*.

pVtx can be unlinked from its edges and the graph using the *VtxClear* function.

[pVtx must have no edges.]

[pVtx must not already be linked to a graph.]

## See Also

*VtxDestroy*, *VtxInit*

## Example

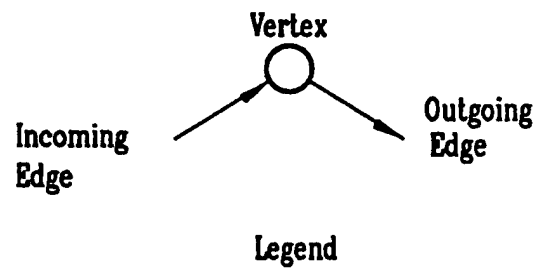
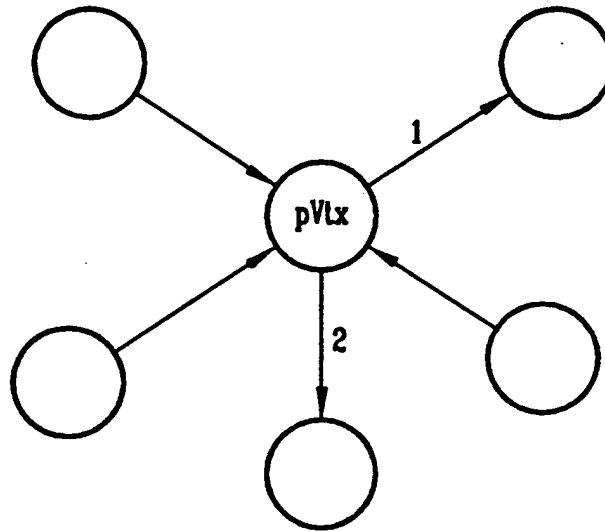
Please refer to class test procedure *TSTGRF.C* for an example of the use of the *VtxDeInit* function.



# VtxCountOut

## Diagram

Outgoing Edges = 2



# VtxCountOut

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
MediumInt    VtxCountOut( pVtx )
PVTX         pVtx;
```

Public Function

A macro is available for this function

## Purpose

The VtxCountOut function returns the number of outgoing edges that are linked to the *Vertex* pVtx.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

The return value from the VtxCountOut function is the number of outgoing edges from the *Vertex* pVtx.

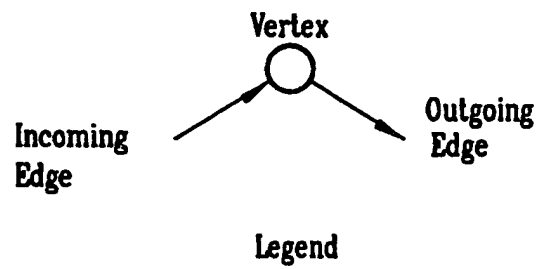
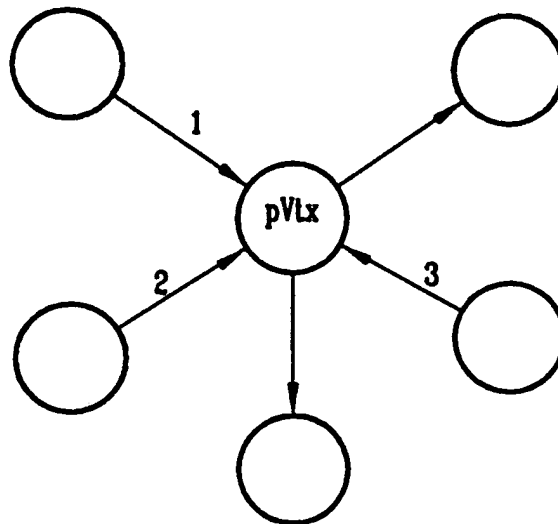
## See Also

VtxCountIn

# VtxCountIn

## Diagram

Incoming Edges = 3



## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
MediumInt    VtxCountIn( pVtx )
PVTX         pVtx;
```

Public Function

A macro is available for this function

## Purpose

The `VtxCountIn` function returns the number of incoming edges that are linked to the *Vertex* `pVtx`.

Parameter	-	Description
-----------	---	-------------

---

<code>pVtx</code>	-	Pointer to a structure of type <i>Vertex</i> .
-------------------	---	--

## Return Value

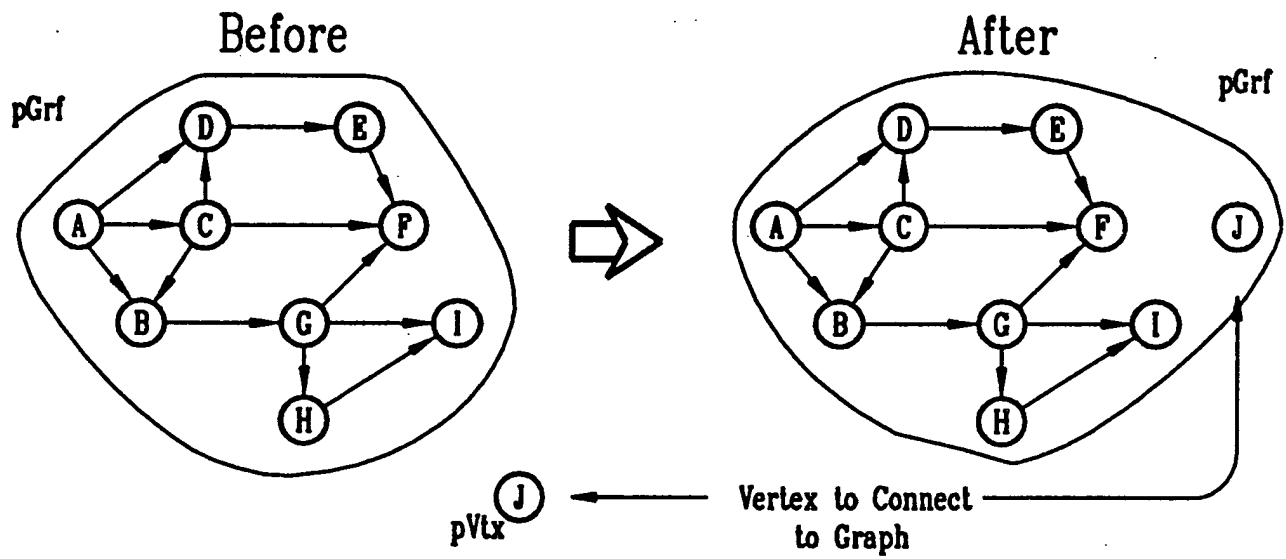
The return value from the `VtxCountIn` function is the number of incoming edges to the *Vertex* `pVtx`.

## See Also

`VtxCountOut`

# VtxConnectToGrf

## Diagram



# VtxConnectToGrf

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxConnectToGrf( pVtx, pGrf )
PVTX      pVtx;
PGRF      pGrf;
```

Public Function

## Purpose

The VtxConnectToGrf links the *Vertex* pVtx to the *Graph* pGrf.

Parameter	-	Description
-----------	---	-------------

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
pGrf	-	Pointer to the <i>Graph</i> structure that the vertex will be linked to.

## Return Value

No return value

## Notes

[pVtx must not already be linked to a graph.]

[pVtx must have no edges.]

## See Also

VtxDisconnectFromGrf

# VtxClear

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
Void      VtxClear( pVtx )
PVTX      pVtx;
```

Public Function

## Purpose

The VtxClear function unlinks the edges (if any) from the *Vertex* pVtx and also unlinks pVtx from the graph. The vertex will be in the same state as it was after being initialized.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

No return value

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxClear function.

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
PDLL      VtxAsOutDll( pVtx )
PVTX      pVtx;
```

Friend Function

## Purpose

The return value from the *VtxAsOutDll* function is a pointer to the *List* structure contained by the *Vertex* *pVtx*. The list contains the outgoing edges to *pVtx*.

Parameter	-	Description
-----------	---	-------------

---

<i>pVtx</i>	-	Pointer to a structure of type <i>Vertex</i> .
-------------	---	--

## Return Value

The return value from the *VtxAsOutDll* function is a pointer to the structure of type *List* contained by the *Vertex* class. The list contains the outgoing edges to *pVtx*.

## See Also

*VtxAsGrfLel*, *VtxAsInDll*, *VtxAsObj*

## Example

Please refer to class test procedure *TSTGRF.C* for an example of the use of the *VtxAsOutDll* function.



# VtxAsObj

---

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
POBJ      VtxAsObj( pVtx )
PVTX      pVtx;
```

Private Function

## Purpose

The VtxAsObj function returns a pointer to the *Object* structure contained by the *Vertex* pVtx.

## Parameter - Description

---

pVtx	-	Pointer to a structure of type <i>Vertex</i> .
------	---	--

## Return Value

The return value from the VtxAsObj function is a pointer to the *Object* structure contained by the *Vertex* class.

## Notes

The *Object* pointer can be used to send a message to the client of the vertex.

## See Also

VtxAsGrfLel, VtxAsInDll, VtxAsOutDll

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxAsObj function.

## Summary

```
#include "cobjects.h"
#include "vtxmac.h"
```

```
PDLL      VtxAsInDll( pVtx )
PVTX      pVtx;
```

Friend Function

## Purpose

The return value from the `VtxAsInDll` function is a pointer to the *List* structure contained by the *Vertex* `pVtx`. The list contains the incoming edges to `pVtx`.

## Parameter - Description

---

<code>pVtx</code>	-	Pointer to a structure of type <i>Vertex</i> .
-------------------	---	--

## Return Value

The return value from the `VtxAsInDll` function is a pointer to the structure of type *List* contained by the *Vertex* class. The list contains the incoming edges to `pVtx`.

## See Also

`VtxAsGrfLel`, `VtxAsObj`, `VtxAsOutDll`

## Example

Please refer to class test procedure `TSTGRF.C` for an example of the use of the `VtxAsInDll` function.

# VtxAsGrfLel

---

## Summary

```
#include "cobjects.h"
```

```
#include "vtxmac.h"
```

```
PLEL      VtxAsGrfLel( pVtx )  
PVTX      pVtx;
```

Friend Function

A macro is available for this function

## Purpose

The VtxAsGrfLel function returns a pointer to the *ListElement* structure contained by the *Vertex* pVtx for linking to the graph.

## Parameter - Description

---

pVtx - Pointer to a structure of type *Vertex*.

## Return Value

The return value from the VtxAsGrfLel function is a pointer to the *ListElement* structure contained by the *Vertex* pVtx.

## See Also

VtxAsInDll, VtxAsObj, VtxAsOutDll

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the VtxAsGrfLel function.

## Class Reference for *Vertex*

**Structure Name:** Vertex  
**Abbreviation:** Vtx  
**Class Type:** Inheritable class

## ***Task***

---

This page is intentionally left blank

# TskRaiseException

---

## Example

Please refer to class test procedure  
TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the  
TskRaiseException function.

# TskRaiseException

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskRaiseException( pTsk, error )
PTSK          pTsk;
MediumInt     error;
```

Public Function

This function is only available as a macro

## Purpose

The TskRaiseException function raises an exception unconditionally for the Task pTsk. The condition type EXT\_FORCED, the file name fileName and the line number in the file lineNo are passed to the function. The signal error can be retrieved by the exception handler.

TskRaiseException can be used to signal special conditions to an exception handler.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
error	-	The signal value.

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## See Also

TskCondition, TskLogCond, TskPreCond, TskPtrCond, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskOnException, TskPropagateExceptionHandler

### See Also

TskCondition, TskLogCond, TskPreCond, TskRaiseException,  
TskMainLogCond, TskMainPreCond, TskMainPtrCond,  
TskMainRaiseException, TskOnException, TskPropagateExceptionHandler

### Example

Please refer to class test procedure  
TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the  
TskPtrCond function.



# TskPtrCond

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskPtrCond( pTsk, error )
PTSK          pTsk;
MediumInt     error;
```

Public Function

This function is only available as a macro

## Purpose

The TskPtrCond function checks the condition flag error and raises an exception if error is zero for the *Task* pTsk. The condition type EXT\_VALID\_PTR, the file name fileName and the line number in the file lineNo are passed to the function.

The TskPtrCond is used throughout C+O to check for illegal pointer values before attempting to execute the function. The production libraries turn off most precondition checking.

TskPtrCond is most frequently used as part of the class pointer asserts found in the header files for each class.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
error	-	If error is equal to zero an exception is triggered.

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## TskPropagateException

---

### Example

Example of using a multiple level exception handler:

```
Void    TskProcess( pTsk )
    PTSK    pTsk;
{
    TskAssert( pTsk );
    .
    .
    /* Set up a second exception handler for special errors */
    if ( TskOnException( pTsk ) ) {
        /* If an exception occurs, program control will jump to */
        /* this IF statement. The code within this IF statement */
        /* will be used to report the exception */
        .
        .
        /* If control should be passed to higher level handler */
        return TskPropagateException( pTsk );
    }
    .
    .
}
```

# TskPropagateException

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskPropagateException( pTsk, useFilter )
PTSK      pTsk;
Bool      useFilter;
```

Public Function

## Purpose

The TskPropagateException function will pass the last exception detected to the next exception handler for the *Task* pTsk. The useFilter flag determines whether the exception filter routine will be called.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
useFilter	-	If True the exception filter routine defined in the TskInit call will be called.

## Return Value

No return value

## Notes

[The exception stack must not be empty for the *Task* pTsk.]

[There must be no recursion of the exception.]

## See Also

TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskCondition, TskOnException

## TskPrintException

---

### Example

Example of printing exception handler message:

```
int    main( argc, argv )
    int    argc;
    char  *argv[];
{
    /* Initialize the Task using the default task TskMain */
    TskDefaultInit( TskMain, argc, argv );

    /* Set up an exception handler using the TskOnException call */
    if ( TskOnException( TskMain ) ) {
        .
        .

        /* Print a message identifying the exception */
        TskPrintException( TskMain );

        /* Exit the task with an error code */
        return TskExit( TskMain, 1 );
    }
    .
    .
}
```

# TskPrintException

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskPrintException( pTsk )
PTSK          pTsk;
```

Public Function

## Purpose

The TskPrintException function prints the description of the exception.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

No return value

### Example

Please refer to class test procedure  
TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the  
TskPreCond function.

# TskPreCond

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskPreCond( pTsk, error )
PTSK          pTsk;
MediumInt     error;
```

### Public Function

This function is only available as a macro

## Purpose

The TskPreCond function checks the condition flag error and raises an exception if error is zero for the *Task* pTsk. The condition type EXT\_PRECONDITION, the file name fileName and the line number in the file lineNo are passed to the function.

The TskPreCond is used throughout C+O to check for illegal parameter values and other preconditions that must hold before attempting to execute the function. The production libraries turn off most precondition checking.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
error	-	If error is equal to zero an exception is triggered.

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## See Also

TskCondition, TskLogCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskOnException, TskPropagateExceptionHandler

## TskPopExceptionHandler

---

### Example

```
Void    TskProcess( pTsk )
    PTKS  pTsk;
{
    TskAssert( pTsk );
    .
    .

    /* Set up a second exception handler for special errors */
    if ( TskOnException( pTsk ) ) {
        .
        .
        .
        TskPrintException( TskMain );
    }
    .
    .

    /* If no exception was detected pop the stack prior to returning */
    TskPopExceptionHandler( pTsk );
}
```



# TskPopExceptionHandler

---

## Summary

```
#include "cobjects.h"  
#include "tskmac.h"
```

```
Void          TskPopExceptionHandler( pTsk )  
PTSK          pTsk;
```

Public Function

## Purpose

The TskPopExceptionHandler function pops to the exception handler stacked below this one for the *Task* pTsk.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

No return value

## Notes

[The exception stack must not be empty for the *Task* pTsk.]

## See Also

TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskOnException, TskPropagateExceptionHandler

# TskOnException

---

## Example

Example of setting up a single level exception handler:

```
int    main( argc, argv )
    int    argc;
    char  *argv[];
{
    /* Initialize the Task using the default task TskMain */
    TskDefaultInit( TskMain, argc, argv );

    /* Set up an exception handler using the TskOnException call */
    if ( TskOnException( TskMain ) ) {
        /* If an exception occurs, program control will jump to */
        /* this IF statement. The code within this IF statement */
        /* will be used to report the exception */
        .
        .
        .

        /* Exit the task with an error code */
        return TskExit( TskMain, 1 );
    }

    /* Call the program code */
    TskProcess( TskMain );

    TskNormalExit( TskMain );
}
```

# TskOnException

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskOnException( pTsk )
PTSK      pTsk;
```

Public Function

This function is only available as a macro

## Purpose

The TskOnException function sets up an exception handler for catching exceptions raised during program execution of *Task* pTsk. The semantics for TskOnException is much like the ANSI C functions setjmp/longjmp. The return value is 0 when the exception handler is first set-up and non-zero if an exception is caught.

Parameter	-	Description
-----------	---	-------------

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

No return value

## Notes

The call to TskOnException should always be within an if statement with no other expressions. The body of the if statement is the exception handler.

Before returning from the function which sets up an exception handler, that function must call TskPopExceptionHandler. The results are undefined if the user fails to do this.

## See Also

TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskPopExceptionHandler, TskCondition, TskPropagateException

# TskNormalExit

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskNormalExit( pTsk )
PTSK          pTsk;
```

Public Function

## Purpose

The TskNormalExit function is equivalent to the C function exit(). The function deinitializes the *Task* pTsk and exits the task with the exit value zero. This function is equivalent to a class deinit function.

## Parameter - Description

---

pTsk - Pointer to a structure of type *Task*.

## Return Value

No return value

## Notes

The TskNormalExit function will be the last function called before returning to the operating system.

## See Also

TskExit, TskExitWithMsg

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskNormalExit function.

# TskMainRaiseException

---

## Example

Please refer to class test procedure  
TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the  
TskMainRaiseException function.

# TskMainRaiseException

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskMainRaiseException( error )
MediumInt error;
```

Public Function

This function is only available as a macro

## Purpose

The TskMainRaiseException function raises an exception unconditionally for the Task TskMain. The condition type EXT\_FORCED, the file name fileName and the line number in the file lineNo are passed to the function. The signal error can be retrieved by the exception handler.

TskMainRaiseException can be used to signal special conditions to an exception handler.

## Parameter - Description

---

error	-	The signal value.
-------	---	-------------------

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## See Also

TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskPopExceptionHandler, TskCondition, TskOnException, TskPropagateException

# TskMainPtrCond

---

## Example

Example of using the TskMainPtrCond function as used by the GrfAssert function in C+O.

```
#define GrfAssert(pGrf) \  
    TskMainPtrCond(pGrf->check==GRF_CHECK_WORD)
```

# TskMainPtrCond

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskMainPtrCond( error )
MediumInt error;
```

### Public Function

This function is only available as a macro

## Purpose

The TskMainPtrCond function checks the condition flag error and raises an exception if error is zero for the Task TskMain. The condition type EXT\_VALID\_PTR, the file name fileName and the line number in the file lineNo are passed to the function.

The TskMainPtrCond is used throughout C+O to check for illegal pointer values before attempting to execute the function. The production libraries turn off most precondition checking.

TskMainPtrCond is most frequently used as part of the class pointer asserts found in the header files for each class.

## Parameter - Description

---

error	-	If error is equal to zero an exception is triggered.
-------	---	--

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## See Also

TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskCondition, TskMainPreCond, TskMainRaiseException, TskPopExceptionHandler, TskOnException, TskPropagateException



## TskMainPreCond

---

### Example

Example of using the TskMainPreCond function as used by the LelTest function in C+O.

```
Void    LelTest( pLel )
        PLEL    pLel;
{
    TskMainPreCond( pLel != NULL ); /* pLel cannot be NULL */
    LelAssert( pLel );

    /* The successor list element must point to pLel */
    if ( pLel->next )
        TskMainPreCond( pLel->next->prev == pLel );

    /* The predecessor list element must point to pLel */
    if ( pLel->prev )
        TskMainPreCond( pLel->prev->next == pLel );
}
```

# TskMainPreCond

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskMainPreCond( error )
MediumInt error;
```

Public Function

This function is only available as a macro

## Purpose

The TskMainPreCond function checks the condition flag error and raises an exception if error is zero for the Task TskMain. The condition type EXT\_PRECONDITION, the file name fileName and the line number in the file lineNo are passed to the function.

The TskMainPreCond is used throughout C+O to check for illegal parameter values and other preconditions that must hold before attempting to execute the function. The production libraries turn off most precondition checking.

## Parameter - Description

---

error	-	If error is equal to zero an exception is trigged.
-------	---	--

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## See Also

TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskCondition, TskMainPtrCond, TskMainRaiseException, TskPopExceptionHandler, TskOnException, TskPropagateException

# TskMainLogCond

---

## Example

Example of using the TskMainLogCond function as used by the GrfClear function in C+O.

```
Void    GrfClear( pGrf )
PGRF    pGrf;
{
    Blk    blk;

    GrfAssert( pGrf );

    BlkInit( &blk, (Method) VtxClear );
    GrfVisitVtxClient( pGrf, 0, &blk );
    BlkDeInit( &blk );

    /* Failed to disconnect all vertices */
    TskMainLogCond( DllIsEmpty( &pGrf->vtxList ) == True );

    /* Failed to disconnect all edges */
    TskMainLogCond( DllIsEmpty( &pGrf->edgList ) == True );

    DpaClear( &pGrf->forwardSort );
    DpaClear( &pGrf->backwardSort );
}
```

# TskMainLogCond

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskMainLogCond( error )
MediumInt error;
```

### Public Function

This function is only available as a macro

## Purpose

The TskLogCond function checks the condition flag error and raises an exception if error is zero for the Task TskMain. The condition type EXT\_LOGIC, the file name fileName and the line number in the file lineNo are passed to the function.

The TskMainLogCond is used throughout C+O to check for program logic errors or function specific conditions which should not arise. The production libraries turn off most logic checking.

Parameter	-	Description
-----------	---	-------------

---

error	-	If error is equal to zero an exception is triggered.
-------	---	--

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## See Also

TskCondition, TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskPopExceptionHandler, TskOnException, TskPropagateException

# TskLogCond

---

## Example

Please refer to class test procedure  
TST1.C,TST2.C,TST3.C,TST4.C,TST5.C,TST6.C for an example of the use of the  
TskLogCond function.

# TskLogCond

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskLogCond( pTsk, error )
PTSK          pTsk;
MediumInt     error;
```

Public Function

This function is only available as a macro

## Purpose

The TskLogCond function checks the condition flag error and raises an exception if error is zero for the *Task* pTsk. The condition type EXT\_LOGIC, the file name fileName and the line number in the file lineNo are passed to the function.

The TskLogCond is used throughout C+O to check for program logic errors or function specific conditions which should not arise. The production libraries turn off most logic checking.

Parameter	-	Description
-----------	---	-------------

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
error	-	If error is equal to zero an exception is triggered.

## Return Value

No return value

## Notes

An exception handler must have been established prior to calling this function. Failure to do so results in program termination and a cryptic error message.

## See Also

TskCondition, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskPopExceptionHandler, TskOnException, TskPropagateException

# TskInit

---

## Example

Please refer to class test procedure

TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskInit function.

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskInit( pTsk, argc, argv, maxNesting, excFilter )
PTSK          pTsk;
MediumInt     argc;
PSTR          *argv;
MediumInt     maxNesting;
ExcFilter     excFilter;
```

Public Function

## Purpose

The TskInit function initializes the *Task* object. The TskInit function should be the first function called when using the *Task* class.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
argc	-	Argument count as passed by main().
argv	-	Argument vector as passed by main().
maxNesting	-	Maximum number of levels for nesting of exception handlers.
excFilter	-	Function to be called just prior to raising exception. If NULL no function is called.

## Return Value

No return value

## Notes

The last function to call when using the *Task* class is TskExit, TskExitWithMsg, or TskNormalExit.

## See Also

TskDefaultInit, TskExit, TskExitWithMsg, TskNormalExit



# TskGetExceptionType

---

## Example

Example of using an exception type:

```
Void    TskProcess( pTsk )
    PTSK    pTsk;
{
    Ext    excType;

    TskAssert( pTsk );
    .
    .
    /* Set up an exception handler for file I/O */
    if ( TskOnException( TskMain ) ) {
        /* If an exception occurs, program control will jump to */
        /* this IF statement. The code within this IF statement */
        /* will be used to report the exception */
        .
        .
        /* Find out the nature of the exception */
        excType = TskGetExceptionType( TskMain );
        if( excType == EXT_FORCED )
            return TskExitWithMsg( pTsk, "I/O Error - program aborted" );
        .
    }
    .
    .
    .
}
```

# TskGetExceptionType

---

## Summary

```
#include "cobjects.h"  
#include "tskmac.h"
```

```
Ext      TskGetExceptionType( pTsk )  
PTSK     pTsk;
```

Public Function

## Purpose

The TskGetExceptionType function returns the exception type.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

The return value from the TskGetExceptionCondition function is the exception type.

## See Also

TskGetExceptionFileName, TskGetExceptionLineNo, TskGetExceptionCondition

# TskGetExceptionLineNo

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
MediumInt    TskGetExceptionLineNo( pTsk )
PTSK         pTsk;
```

Public Function

## Purpose

The TskGetExceptionLineNo function returns the line number of the file where the exception was generated for the *Task* pTsk.

## Parameter - Description

---

pTsk - Pointer to a structure of type *Task*.

## Return Value

The return value from the TskGetExceptionLineNo function is the line number in a C source code file where the exception occurred.

## See Also

TskGetExceptionFileName, TskGetExceptionCondition, TskGetExceptionType

## Example

Please refer to class test procedure  
TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the  
TskGetExceptionLineNo function.

# TskGetExceptionFileName

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
PSTR      TskGetExceptionFileName( pTsk )
PTSK      pTsk;
```

Public Function

## Purpose

The TskGetExceptionFileName function returns the filename where the exception was generated for the *Task* pTsk.

## Parameter - Description

---

pTsk - Pointer to a structure of type *Task*.

## Return Value

The return value from the TskGetExceptionFileName function is the name of the C source code file where the exception occurred.

## See Also

TskGetExceptionLineNo, TskGetExceptionCondition, TskGetExceptionType

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskGetExceptionFileName function.

# TskGetExceptionCondition

---

## Summary

```
#include "cobjects.h"  
#include "tskmac.h"
```

```
MediumInt  TskGetExceptionCondition( pTsk )  
PTSK      pTsk;
```

Public Function

## Purpose

The TskGetExceptionCondition function returns the exception condition error value.

Parameter	-	Description
-----------	---	-------------

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

The return value from the TskGetExceptionCondition function is the exception condition error value.

## See Also

TskGetExceptionFileName, TskGetExceptionLineNo, TskGetExceptionType

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskGetExceptionCondition function.

# TskGetArgv

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
PSTR      TskGetArgv( pTsk )
PTSK      pTsk;
```

Public Function

## Purpose

The TskGetArgv function returns the argument vector as passed by main() for the *Task* pTsk.

Parameter	-	Description
-----------	---	-------------

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

The return value from the TskGetArgv function is a pointer to a *String* pointer.

## See Also

TskGetArgc

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskGetArgv function.

# TskGetArgc

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
MediumInt    TskGetArgc( pTsk )
PTSK         pTsk;
```

Public Function

## Purpose

The TskGetArgc function returns the argument count as supplied by main() for the *Task* pTsk.

Parameter	-	Description
-----------	---	-------------

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

## See Also

TskGetArgv

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskGetArgc function.

# TskExitWithMsg

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskExitWithMsg( pTsk, msg )
PTSK          pTsk;
PSTR          msg;
```

Public Function

## Purpose

The TskExitWthMsg function is equivalent to the C function exit() with a printf(msg) statement immediately before it. The function deinitializes the Task pTsk and exits the task with an exit value of 1. This function is equivalent to a class deinit function.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type Task.
msg	-	Message string to print to the terminal using the C function printf(msg).

## Return Value

No return value

## Notes

The TskExitWithMsg function will be the last function called before returning to the operating system.

## See Also

TskExit, TskNormalExit

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskExitWithMsg function.



# TskExit

---

## Summary

```
#include "cobjects.h"
```

```
#include "tskmac.h"
```

```
Void          TskExit( pTsk, exitVal )  
PTSK          pTsk;  
MediumInt     exitVal;
```

Public Function

## Purpose

The TskExit function is equivalent to the C function exit(). The function deinitializes the *Task* pTsk and exits the task with the exit value exitVal. This function is equivalent to a class deinit function.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
exitVal	-	Exit value to pass to the C exit() function.

## Return Value

No return value

## Notes

The TskExit function will be the last function called before returning to the operating system.

## See Also

TskExitWithMsg, TskNormalExit

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskExit function.

# TskDefaultInit

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void          TskDefaultInit( pTsk, argc, argv )
PTSK          pTsk;
MediumInt     argc;
PSTR          argv[];
```

Public Function

## Purpose

The TskDefaultInit function initializes the *Task* object. The TskDefaultInit function should be the first function called when using the *Task* class. The user can also use the TskInit function which requires additional arguments.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
argc	-	Argument count as passed by main().
argv	-	Argument vector as passed by main().

## Return Value

No return value

## Notes

The last function to call when using the *Task* class is TskExit, TskExitWithMsg, or TskNormalExit.

## See Also

TskExit, TskExitWithMsg, TskInit, TskNormalExit

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskDefaultInit function.

# TskDeInit

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskDeInit( pTsk )
PTSK      pTsk;
```

Public Function

## Purpose

The TskDeInit function deinitializes the *Task* object. The TskDeInit function should be the last function called when using the *Task* class.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
------	---	--

## Return Value

No return value

## Notes

The first function to call when using the *Task* class is TskInit.

## See Also

TskDefaultInit, TskInit

## Example

Please refer to class test procedure TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskDeInit function.

# TskCondition

---

## Notes

An exception handler must have been established prior to calling this function.

[The exception stack must not be empty for the *Task* pTsk.]

[There must be no recursion of the exception.]

[The condition category must be valid.]

## See Also

TskLogCond, TskPreCond, TskPtrCond, TskRaiseException, TskMainLogCond, TskMainPreCond, TskMainPtrCond, TskMainRaiseException, TskPopExceptionHandler, TskOnException

## Example

Please refer to class test procedure

TST1.C, TST2.C, TST3.C, TST4.C, TST5.C, TST6.C for an example of the use of the TskCondition function.

# TskCondition

---

## Summary

```
#include "cobjects.h"
#include "tskmac.h"
```

```
Void      TskCondition( pTsk, type, error, fileName, lineNo )
PTSK      pTsk;
Ext       type;
MediumInt error;
PSTR      fileName;
MediumInt lineNo;
```

Public Function

## Purpose

The TskCondition function checks the condition flag error and raises an exception if error is zero for the *Task* pTsk. The category of the condition type, the file name fileName and the line number in the file lineNo are passed to the function.

## Parameter - Description

---

pTsk	-	Pointer to a structure of type <i>Task</i> .
type	-	Condition category of exception. This will be defined by the choice of <i>Task</i> exception macro.
error	-	If error is equal to zero an exception is triggered. If type is EXT_FORCED any value of error will trigger exception.
fileName	-	Name of file where exception occurred.
lineNo	-	Line number in file where exception occurred.

Condition categories:

EXT\_PRECONDITION

EXT\_LOGIC

EXT\_VALID\_PTR

EXT\_FORCED

Preconditions for execution, examples; function arguments, ranges, links.

Conditions during execution of code, examples; logic errors, out of bounds.

Used for checking the check word of objects.

Forced exception, value of error overridden, examples; error during read/write.

## Return Value

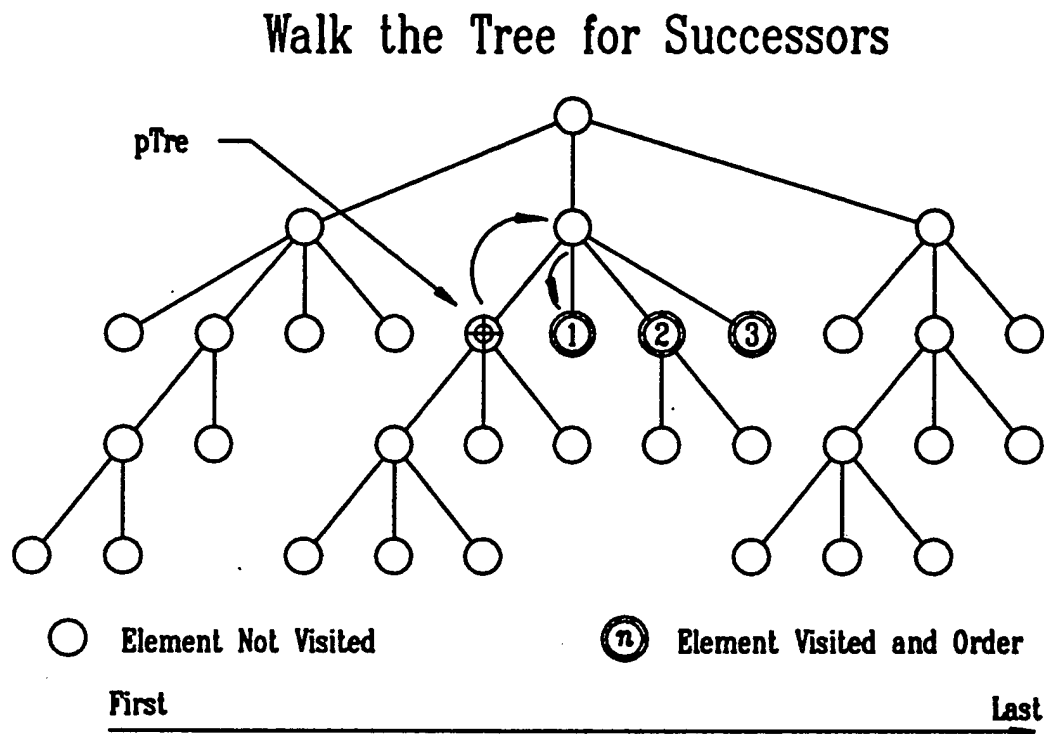
No return value

## Class Reference for *Task*

**Structure Name:** Task  
**Abbreviation:** Tsk  
**Class Type:** Primitive Class

---

### Diagram



# TreVisitSuccessors

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitSuccessors( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitSuccessors function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked in a forward direction for all successors and starts with the tree node that is the next sibling to pTre.

The *Block* pBlk contains the tree function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

## See Also

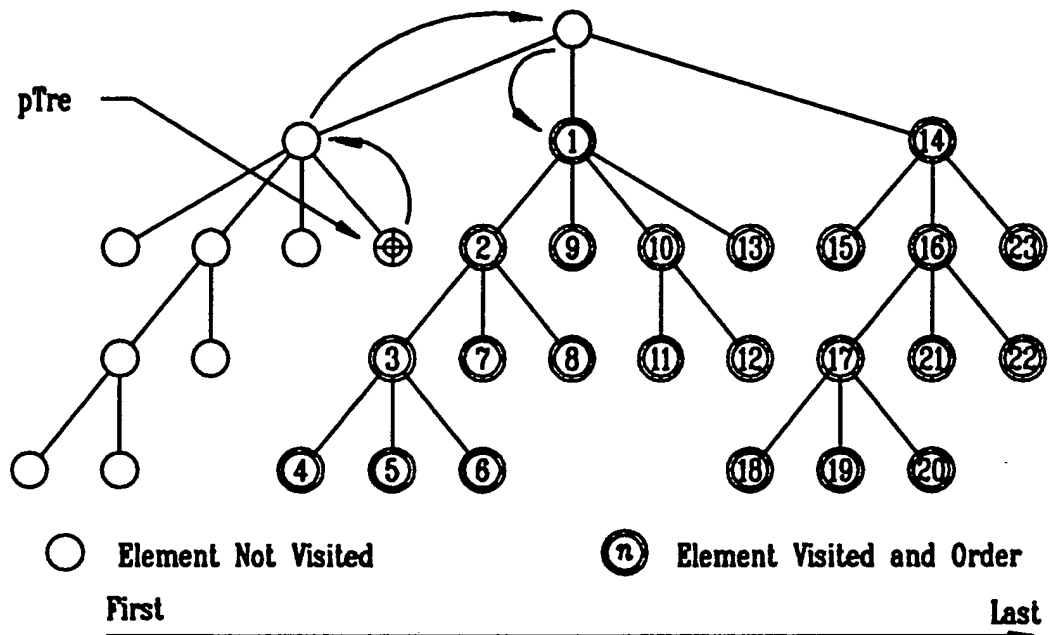
TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder, TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd, TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder, TreVisitDescInOrderBwd, TreVisit Leaves, TreVisitParents, TreVisitRange



# TreVisitSuccPreOrder

## Diagram

Walk the Tree for all successors in Pre-order



# TreVisitSuccPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitSuccPreOrder( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitSuccPreOrder function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked starting with the pre-order successor to pTre and visits all successors of pTre.

The *Block* pBlk contains the tree function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The PreOrder successor to this node is first visited.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

## See Also

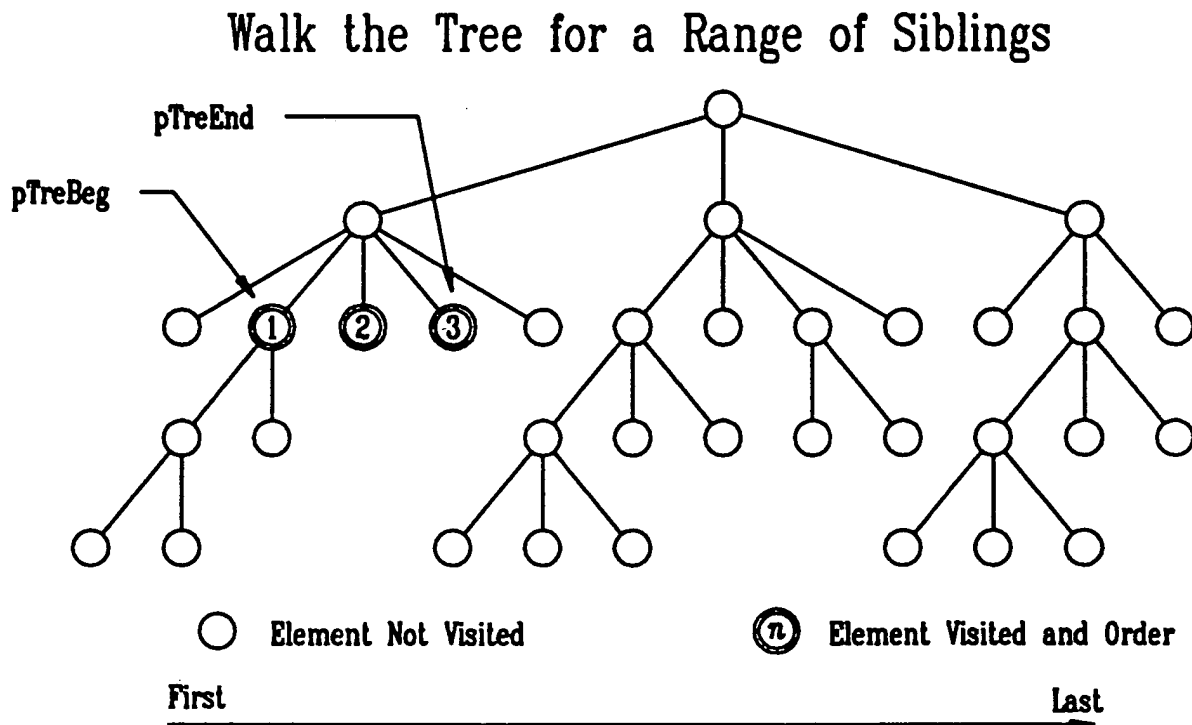
TreVisitBranchInOrder, TreVisitPreOrder, TreVisitChildren,  
TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder,  
TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitParents, TreVisitRange,  
TreVisitSuccessors

# TreVisitRange

## See Als

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder, TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd, TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder, TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitParents, TreVisitSuccessors

## Diagram



## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
MediumInt    TreVisitRange( pTreBeg, pTreEnd, pBlk )
PTRE         pTreBeg;
PTRE         pTreEnd;
PBLK         pBlk;
```

Private Function

## Purpose

The `TreVisitRange` function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked for a range of sibling nodes, `pTreBeg` through `pTreEnd`. The siblings are visited in a forward direction.

The *Block* `pBlk` contains the tree function and an optional list of arguments.

A range of tree nodes is defined as a beginning tree node and an ending tree node. The beginning node can equal the ending node. See *Tree* class section on range definition for more details.

Parameter	-	Description
-----------	---	-------------

---

<code>pTreBeg</code>	-	Pointer to a structure of type <i>Tree</i> . This is the first tree node in a range.
<code>pTreEnd</code>	-	Pointer to a structure of type <i>Tree</i> . This is the last tree node in a range.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

The return value from the `TreVisitRange` function is the number of times tree nodes are walkd.

## Notes

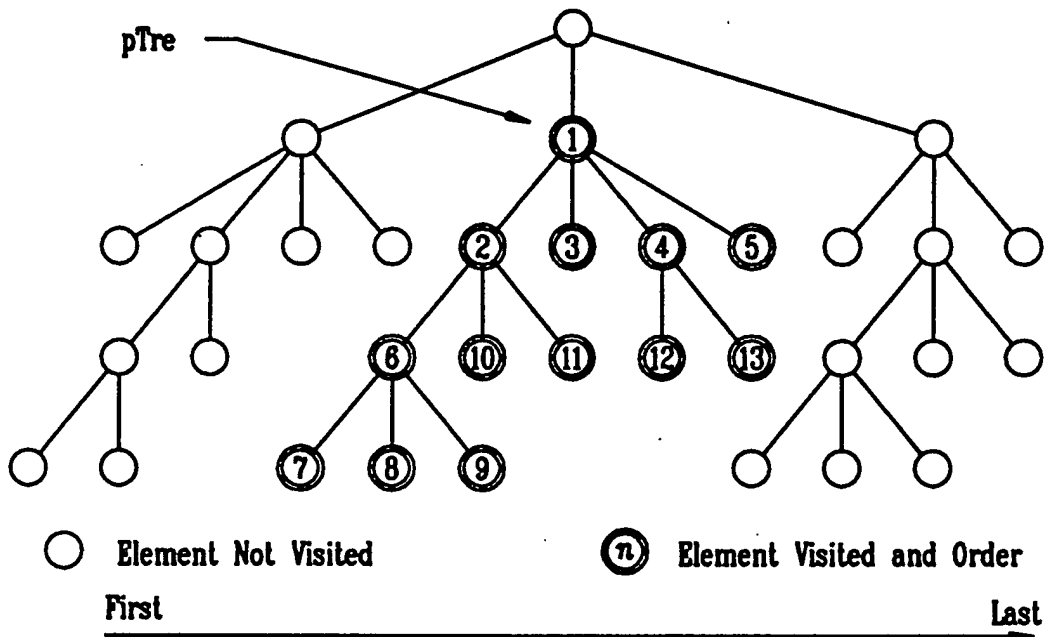
The tree function may return a value but it is ignored.

[If `pTreBeg` does not equal `pTreEnd` then `pTreBeg` must precede `pTreEnd`.]

# TreVisitPreOrder

## Diagram

Walk the Tree in Pre-order



# TreVisitPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitPreOrder( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitPreOrder function walks the *Tree* and calls a *Tree* function for each tree node visited. The walk visits pTre first and calls TreVisitPreOrder recursively for each of its children.

The *Block* pBlk contains the tree function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

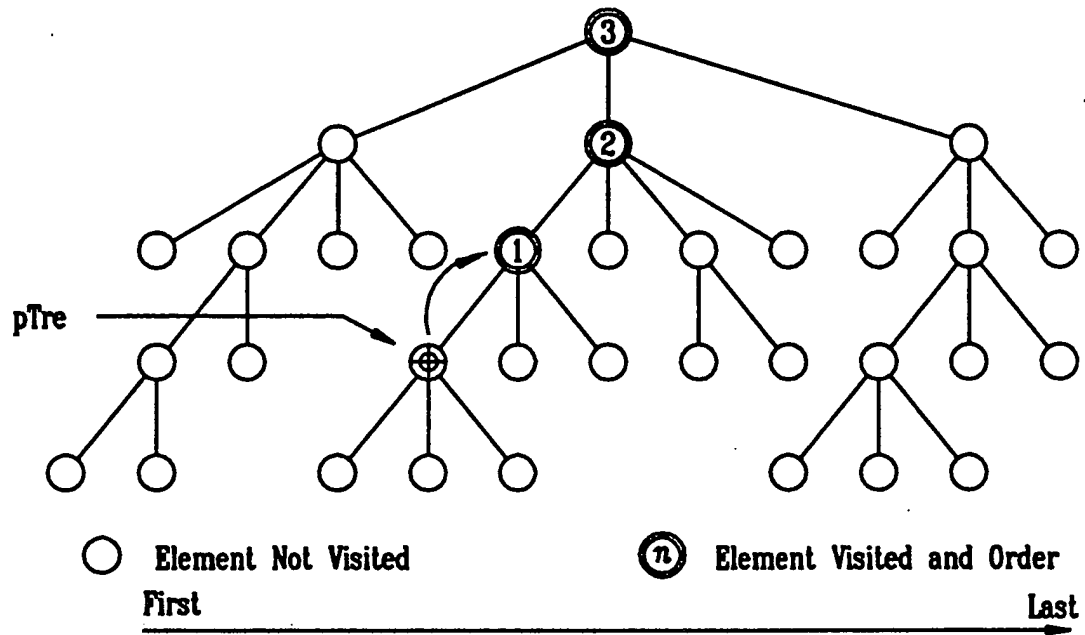
## See Also

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitChildren,  
TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder,  
TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitParents, TreVisitRange,  
TreVisitSuccessors

# TreVisitParents

## Diagram

Walk the Tree Nearest Parents First



# TreVisitParents

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitParents( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitParents function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked by calling TreVisitParents recursively for the parent of pTre.

The *Block* pBlk contains the tree function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

## See Also

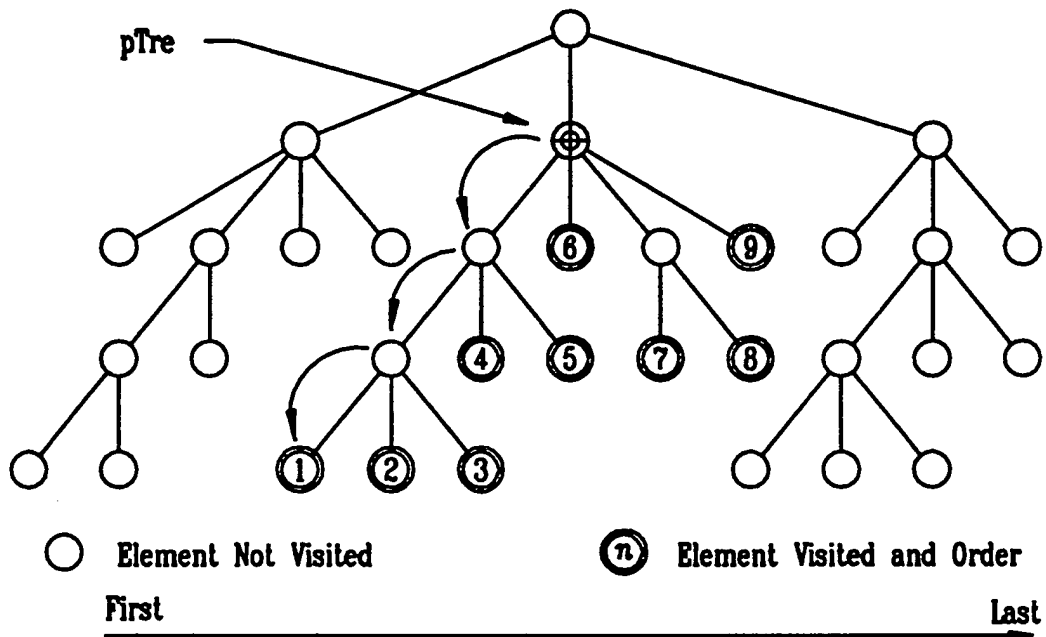
TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder, TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd, TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder, TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitRange, TreVisitSuccessors



# TreVisitLeaves

## Diagram

### Walk the Tree for Leaves



# TreVisitLeaves

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitLeaves( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitLeaves function walks the *Tree* and calls the *Tree* function for each tree node visited. The function calls TreVisitLeaves recursively for each of its children. The nodes are visited if they are leaves (have no children).

The *Block* pBlk contains the tree function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

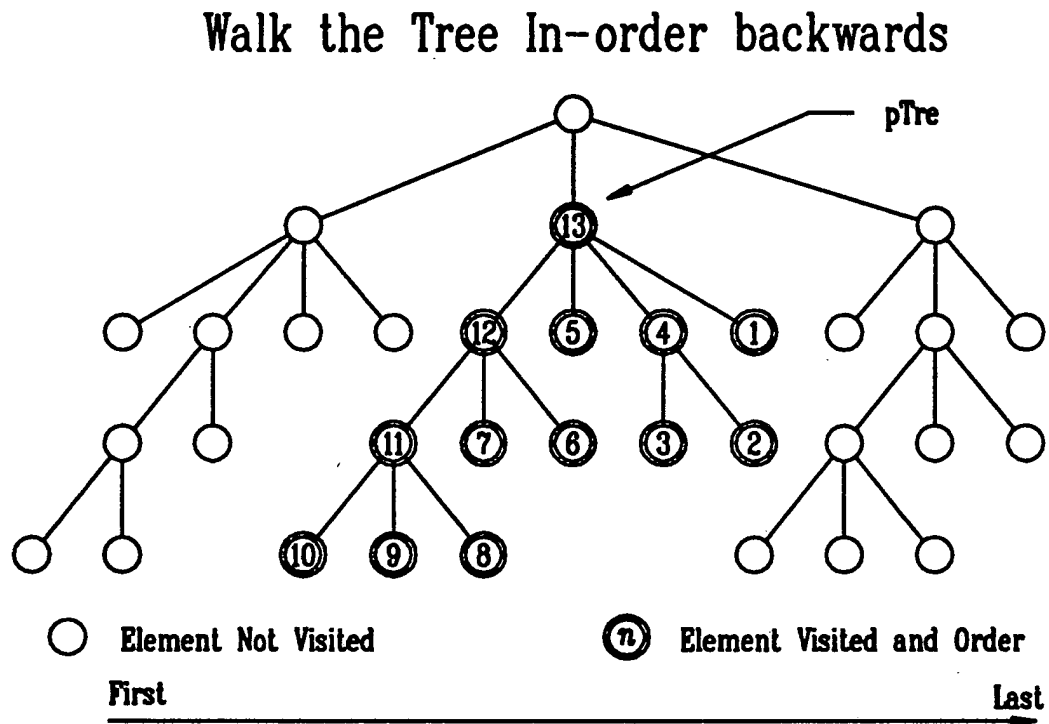
The tree function may return a value but it is ignored.

## See Also

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder,  
TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder,  
TreVisitDescInOrderBwd, TreVisitParents, TreVisitRange, TreVisitSuccessors

# TreVisitInOrderBwd

## Diagram



# TreVisitInOrderBwd

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitInOrderBwd( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The `TreVisitInOrderBwd` function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked by calling `TreVisitInOrderBwd` recursively for the children of `pTre`, and then visiting `pTre` itself. The nodes are visited in a backward direction.

The *Block* `pBlk` contains the tree function and an optional list of arguments.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

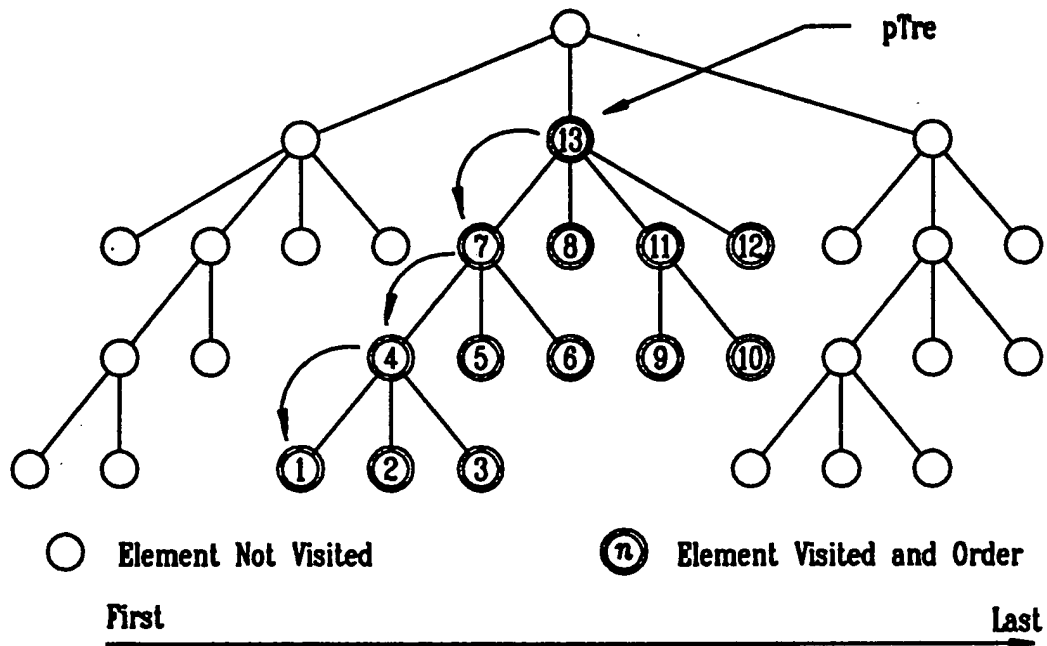
The tree function may return a value but it is ignored.

## See Also

`TreVisitSuccPreOrder`, `TreVisitBranchInOrder`, `TreVisitPreOrder`,  
`TreVisitChildren`, `TreVisitChildrenBwd`, `TreVisitInOrder`,  
`TreVisitDescBranchInOrder`, `TreVisitDescPreOrder`, `TreVisitDescInOrder`,  
`TreVisitDescInOrderBwd`, `TreVisitLeaves`, `TreVisitParents`, `TreVisitRange`,  
`TreVisitSuccessors`

### Diagram

## Walk the Tree In-order



## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitInOrder( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitInOrder function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked by calling TreVisitInOrder recursively for the children of pTre, and then visiting pTre itself. The nodes are visited in a forward direction.

The *Block* pBlk contains the tree function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

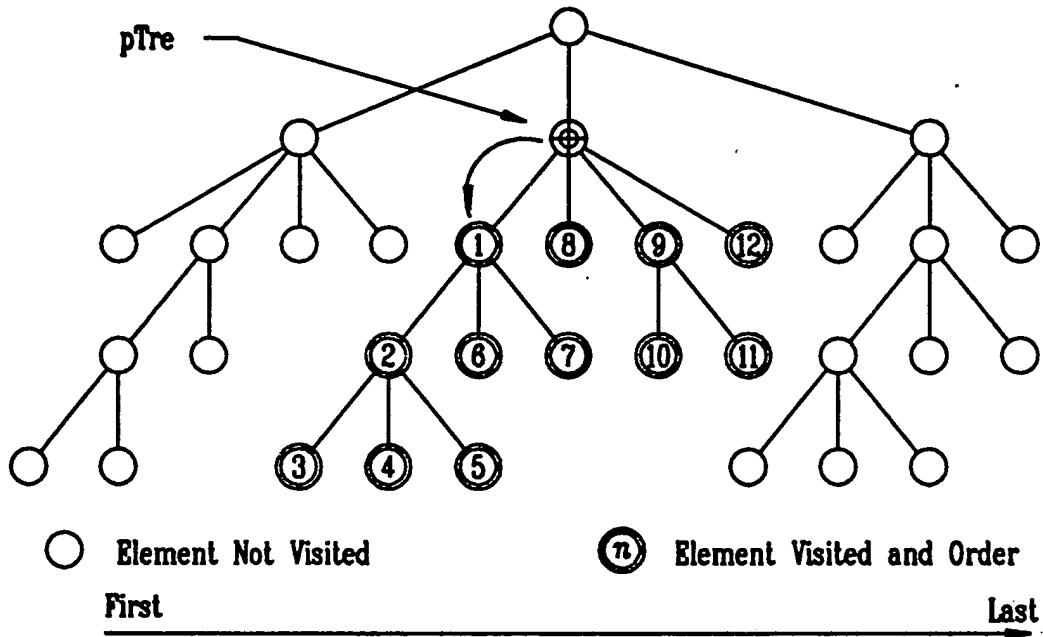
## See Also

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder,  
TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder,  
TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitParents, TreVisitRange,  
TreVisitSuccessors

# TreVisitDescPreOrder

## Diagram

Walk the Tree Descendents in Pre-order



# TreVisitDescPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitDescPreOrder( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitDescPreOrder function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked by calling TreVisitPreOrder for each of its children.

The *Block* pBlk contains the tree function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

## See Also

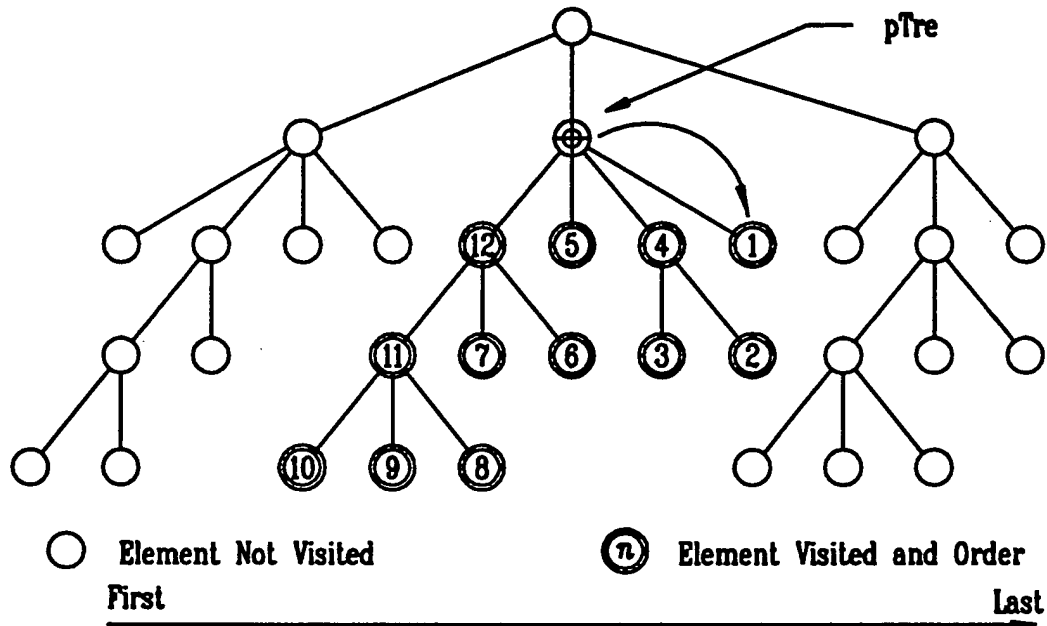
TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder, TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd, TreVisitDescBranchInOrder, TreVisitDescInOrder, TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitParents, TreVisitRange, TreVisitSuccessors



# TreVisitDescInOrderBwd

## Diagram

Walk the Tree Descendents In-order backwards



# TreVisitDescInOrderBwd

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitDescInOrderBwd( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitDescInOrderBwd function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked by calling TreVisitInOrderBwd for the children of pTre. The nodes are visited in a backward direction.

The *Block* pBlk contains the tree function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

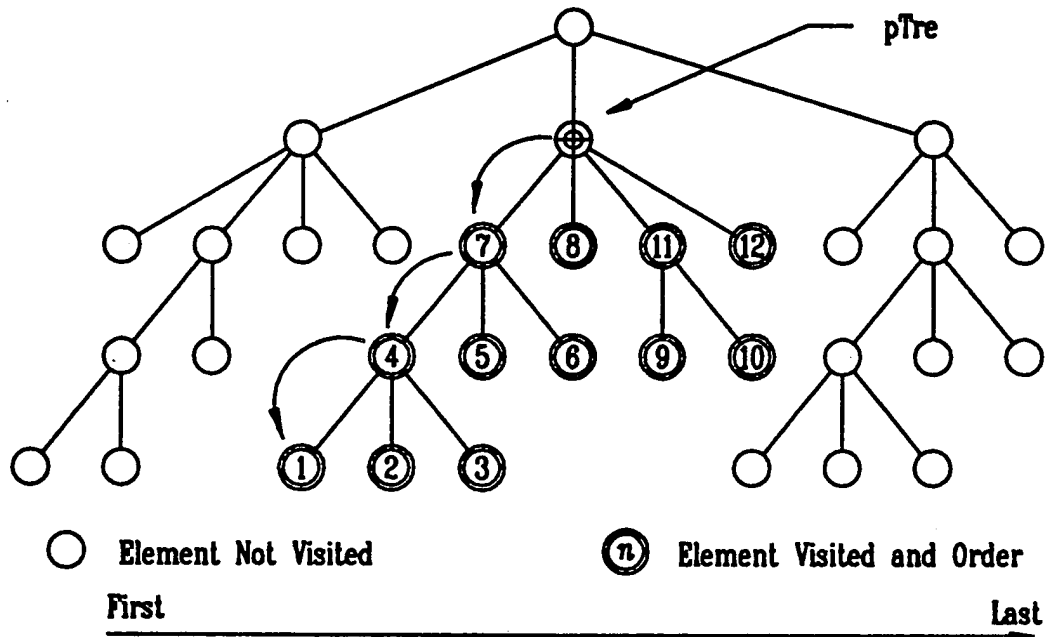
## See Also

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder,  
TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder,  
TreVisitLeaves, TreVisitParents, TreVisitRange, TreVisitSuccessors

# TreVisitDescInOrder

## Diagram

Walk the Tree Descendents In-order



# TreVisitDescInOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitDescInOrder( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitDescInOrder function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked by calling TreVisitInOrder for the children of pTre. The nodes are visited in a forward direction.

The *Block* pBlk contains the tree function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

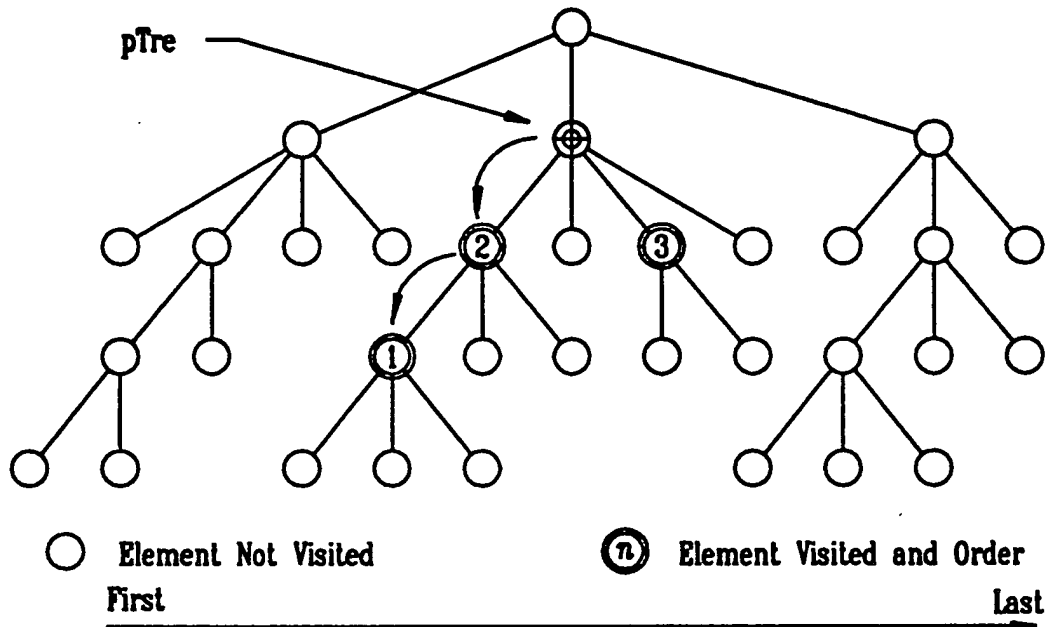
## See Also

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder,  
TreVisitChildren, TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrderBwd,  
TreVisitLeaves, TreVisitParents, TreVisitRange, TreVisitSuccessors

# TreVisitDescBranchInOrder

## Diagram

Walk the Tree Descendent branches In-order



# TreVisitDescBranchInOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitDescBranchInOrder( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The `TreVisitDescBranchInOrder` function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked by calling `TreVisitBranchInOrder` for each of its children. The nodes are visited if they are branches (have children).

The *Block* `pBlk` contains the tree function and an optional list of arguments.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

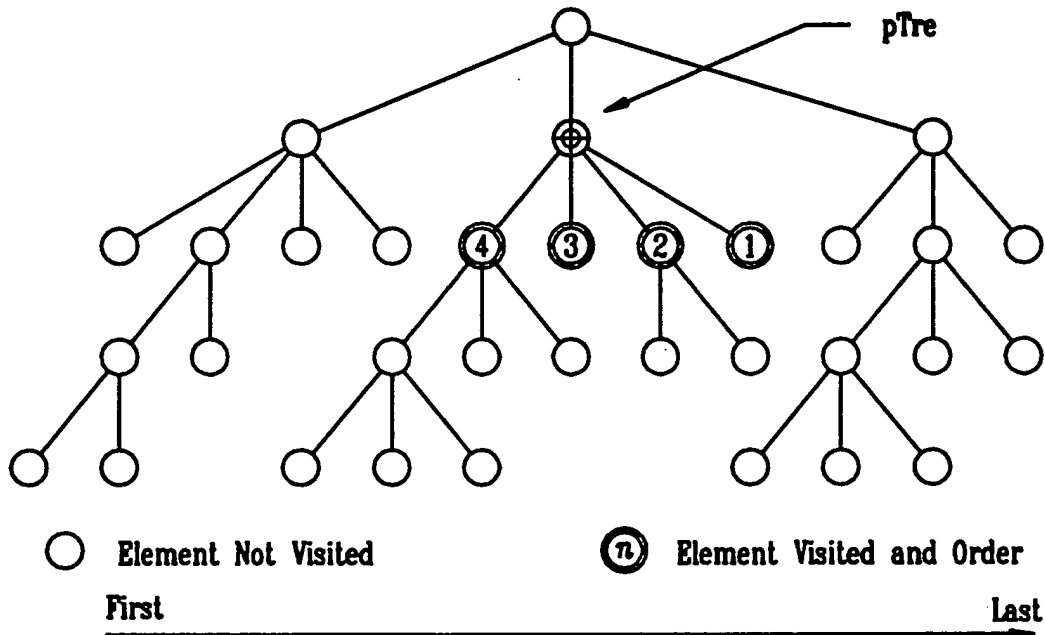
## See Also

`TreVisitSuccPreOrder`, `TreVisitBranchInOrder`, `TreVisitPreOrder`,  
`TreVisitChildren`, `TreVisitChildrenBwd`, `TreVisitInOrder`, `TreVisitInOrderBwd`,  
`TreVisitDescPreOrder`, `TreVisitDescInOrder`, `TreVisitDescInOrderBwd`,  
`TreVisitLeaves`, `TreVisitParents`, `TreVisitRange`, `TreVisitSuccessors`

# TreVisitChildrenBwd

## Diagram

Walk the Tree for all Children Backwards



# TreVisitChildrenBwd

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitChildrenBwd( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitChildrenBwd function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked for each of the children of the *Tree* pTre starting with its last child and ending with its first child.

The *Block* pBlk contains the tree function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

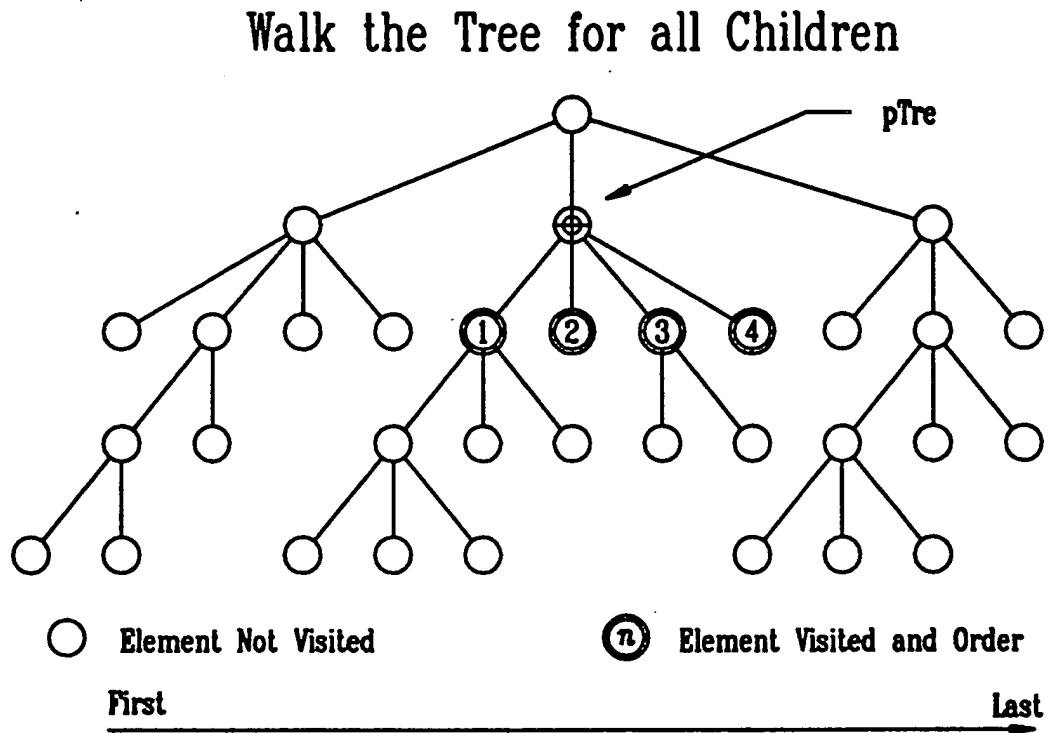
## See Also

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder,  
TreVisitChildren, TreVisitInOrder, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder,  
TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitParents, TreVisitRange,  
TreVisitSuccessors



# TreVisitChildren

## Diagram



# TreVisitChildren

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitChildren( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The TreVisitChildren function walks the *Tree* and calls a *Tree* function for each tree node visited. The *Tree* is walked for each of the children of the *Tree* pTre starting with its first child and ending with its last child.

The *Block* pBlk contains the tree function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

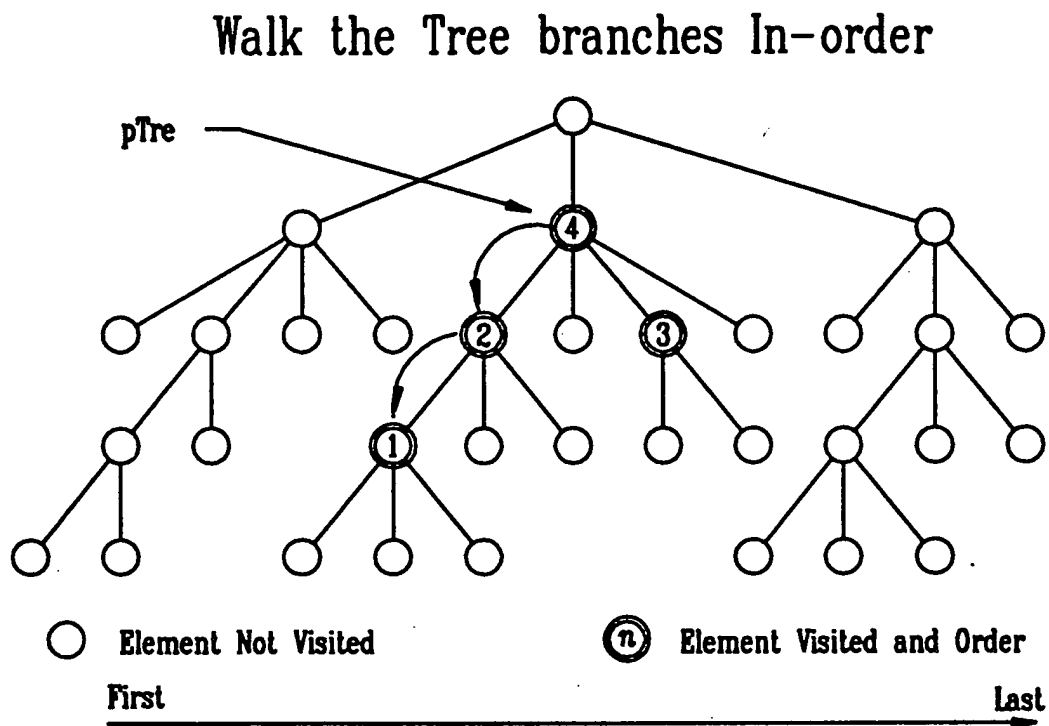
The tree function may return a value but it is ignored.

## See Also

TreVisitSuccPreOrder, TreVisitBranchInOrder, TreVisitPreOrder,  
TreVisitChildrenBwd, TreVisitInOrder, TreVisitInOrderBwd,  
TreVisitDescBranchInOrder, TreVisitDescPreOrder, TreVisitDescInOrder,  
TreVisitDescInOrderBwd, TreVisitLeaves, TreVisitParents, TreVisitRange,  
TreVisitSuccessors

# TreVisitBranchInOrder

## Diagram



# TreVisitBranchInOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreVisitBranchInOrder( pTre, pBlk )
PTRE      pTre;
PBLK      pBlk;
```

Private Function

## Purpose

The `TreVisitBranchInOrder` function walks a *Tree* and calls a *Tree* function for each tree node visited. The tree is walked by calling `TreVisitBranchInOrder` recursively for each of its children, and then visiting `pTre` itself. The nodes are visited if they are branches (have children).

The *Block* `pBlk` contains the tree function and an optional list of arguments.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the tree function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The tree function may return a value but it is ignored.

## See Also

`TreVisitSuccPreOrder`, `TreVisitPreOrder`, `TreVisitChildren`,  
`TreVisitChildrenBwd`, `TreVisitInOrder`, `TreVisitInOrderBwd`,  
`TreVisitDescBranchInOrder`, `TreVisitDescPreOrder`, `TreVisitDescInOrder`,  
`TreVisitDescInOrderBwd`, `TreVisitLeaves`, `TreVisitParents`, `TreVisitRange`,  
`TreVisitSuccessors`

# TreSendDestroy

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreSendDestroy( pTre )
PTRE      pTre;
```

Public Function

## Purpose

The TreSendDestroy function sends a message to the client of the *Tree* pTre asking it to destroy the tree node. The *Tree* client function will receive this message and should deinitialize or destroy the edge. This message function should be included in the *Tree* client message array.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

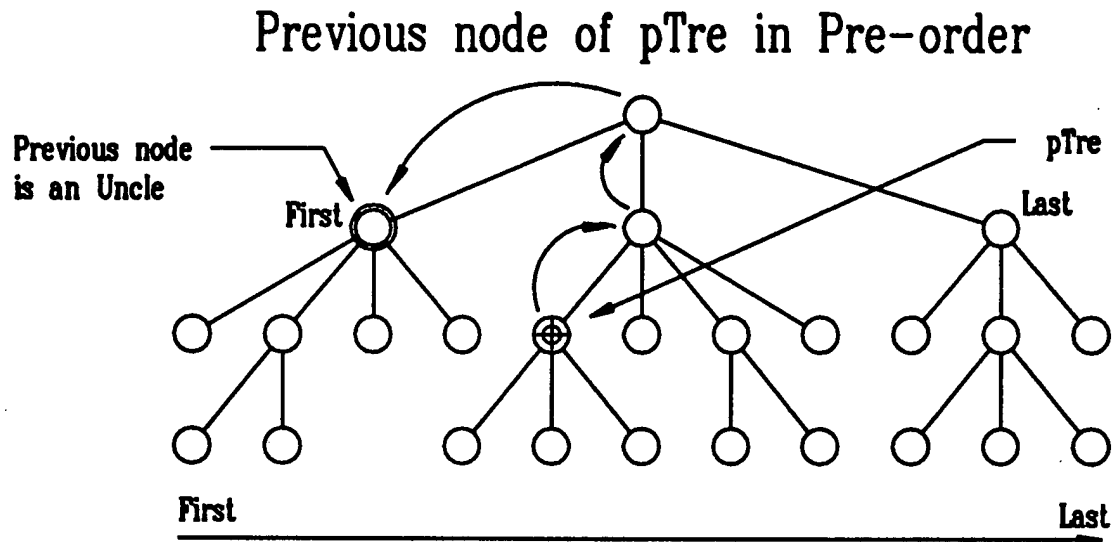
No return value

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreSendDestroy function.

# TrePrevPreOrder

## Diagram



# TrePrevPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PTRE      TrePrevPreOrder( pTre )
PTRE      pTre;
```

Private Function

## Purpose

The TrePrevPreOrder function returns the client of the pre-order predecessor the *Tree* pTre. The pre-order predecessor is the previous sibling of pTre. If pTre has no previous sibling the function TreClientPrevPreOrder is called recursively for the parent of pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

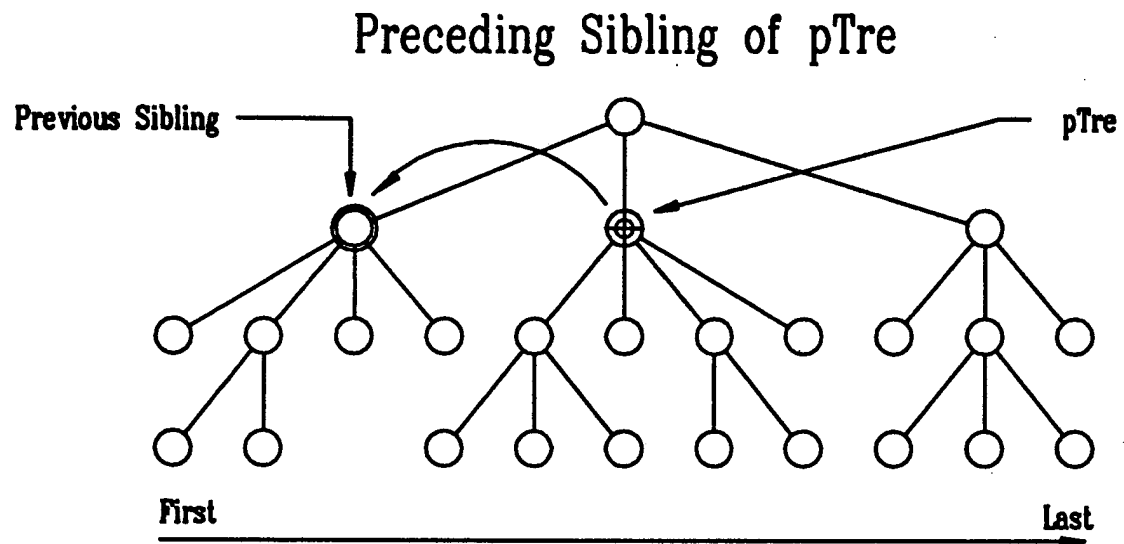
## Return Value

The return value from the TrePrevPreOrder function is a pointer to the pre-order predecessor of the *Tree* pTre. NULL is returned if no node exists.

## See Also

TreFirstChild, TreLastChild, TreLastLeaf, TreNext, TreNextUncle, TreNextPreOrder, TreParent, TrePrev

Diagram





# TrePrev

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PTRE      TrePrev( pTre )
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The TrePrev function returns the preceding sibling of the *Tree* pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TrePrev function is a pointer to the preceding sibling of the *Tree* pTre. NULL is returned if no previous sibling exists.

## See Also

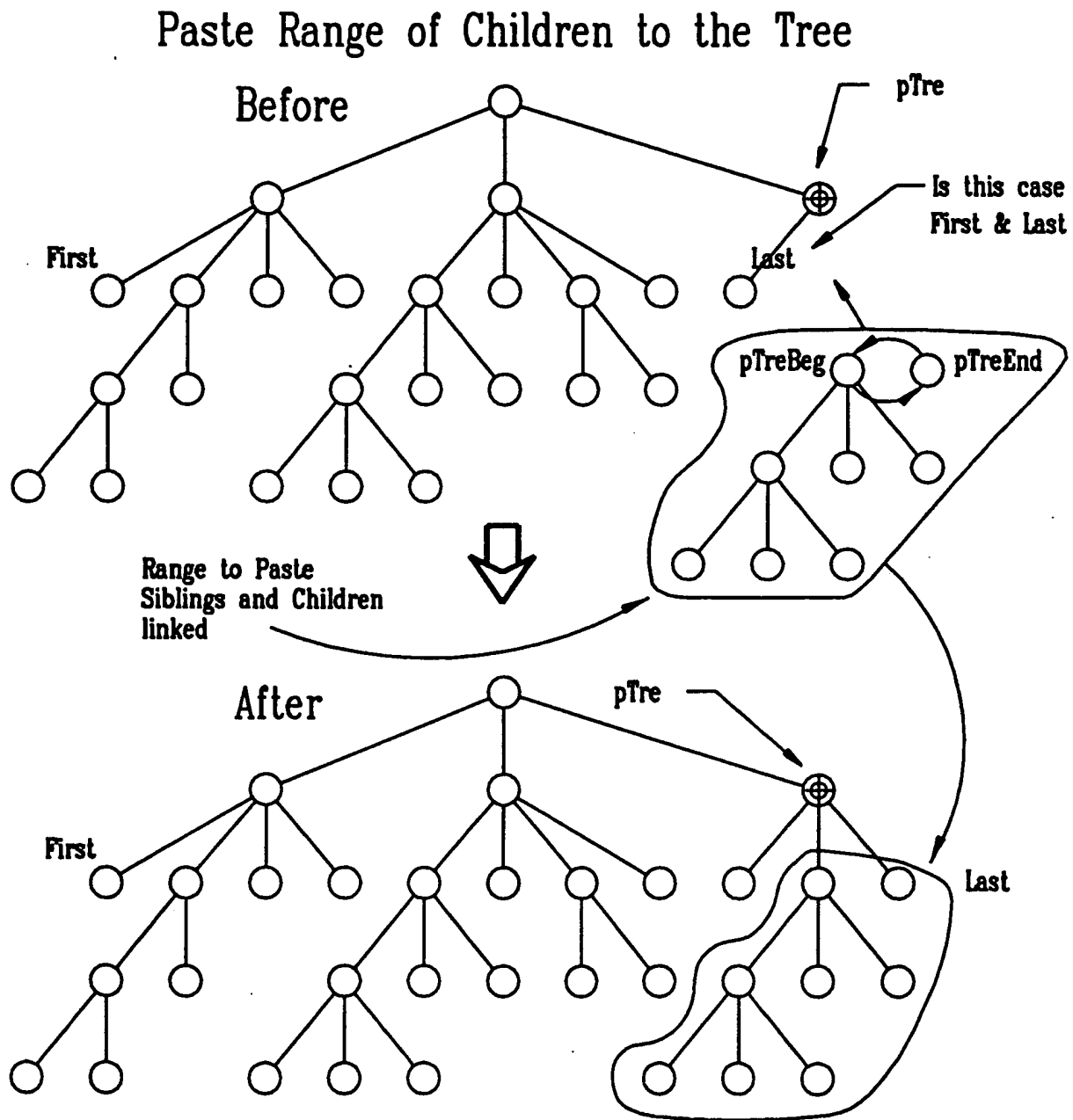
TreFirstChild, TreLastChild, TreLastLeaf, TreNext, TreNextUncle, TreNextPreOrder, TreParent, TrePrevPreOrder

## TrePasteRangeLastChild

### See Also

TreCutRange, TreCutChildren, TrePasteRangeFirstChild,  
TrePasteRangeAfterSibling, TrePasteRangeBeforeSibling

### Diagram



# TrePasteRangeLastChild

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TrePasteRangeLastChild( pTre, pTreBeg, pTreEnd )
PTRE      pTre;
PTRE      pTreBeg;
PTRE      pTreEnd;
```

Public Function

A macro is available for this function

## Purpose

The `TrePasteRangeLastChild` function links a range of the *Tree* nodes `pTreBeg` through `pTreEnd` to the *Tree*. The range is linked so that `pTreEnd` is the last child of `pTre`.

The tree node range must have been previously cut from a tree.

A range of tree nodes is defined as a beginning tree node and an ending tree node. The beginning node can equal the ending node. See *Tree* class section on range definition for more details.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The appended tree node(s) become the last child(ren) of this node.
<code>pTreBeg</code>	-	Pointer to a structure of type <i>Tree</i> . This is the first tree node to paste into the tree as a child of <code>pTre</code> .
<code>pTreEnd</code>	-	Pointer to a structure of type <i>Tree</i> . This is the last tree node to paste into the tree as a child of <code>pTre</code> .

## Return Value

No return value

## Notes

`pTreBeg` must have a set of successors one of which is `pTreEnd`.

[If `pTreBeg` does not equal `pTreEnd` then `pTreBeg` must precede `pTreEnd`.]

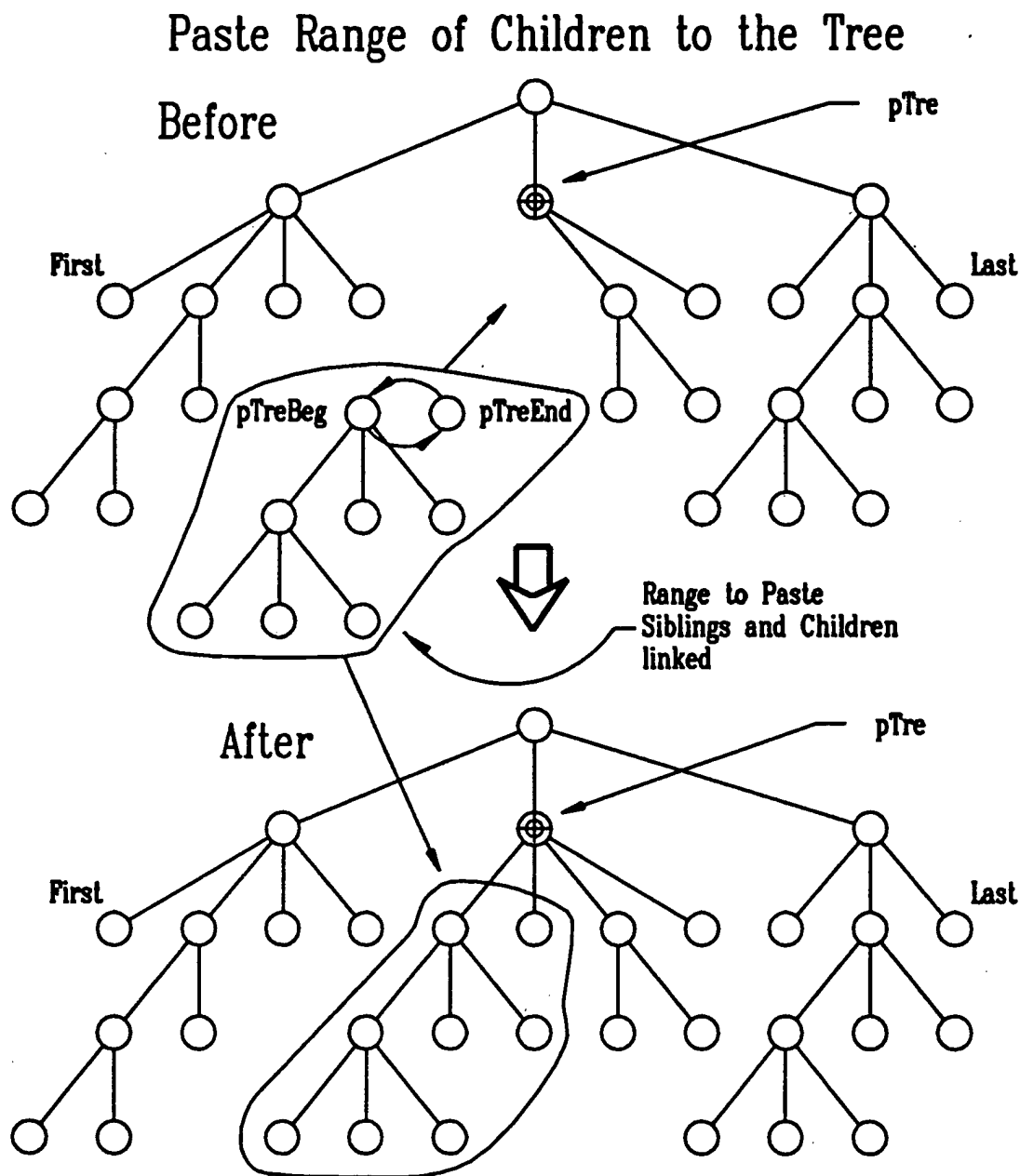
[`pTreBeg` must be the root node.]

## TrePasteRangeFirstChild

### See Also

TreCutRange, TreCutChildren, TrePasteRangeLastChild,  
TrePasteRangeAfterSibling, TrePasteRangeBeforeSibling

### Diagram



# TrePasteRangeFirstChild

---

## Summary

```
#include "cobjects.h"
```

```
#include "tremac.h"
```

```
Void      TrePasteRangeFirstChild( pTre, pTreBeg, pTreEnd )
PTRE      pTre;
PTRE      pTreBeg;
PTRE      pTreEnd;
```

Public Function

A macro is available for this function

## Purpose

The `TrePasteRangeFirstChild` function links a range of the *Tree* nodes `pTreBeg` through `pTreEnd` to the *Tree* `pTre`. The range is linked so that `pTreBeg` is the first child of `pTre`.

The tree node range must have been previously cut from a tree.

A range of tree nodes is defined as a beginning tree node and an ending tree node. The beginning node can equal the ending node. See *Tree* class section on range definition for more details.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The inserted tree node(s) become the first child(ren) of this node.
<code>pTreBeg</code>	-	Pointer to a structure of type <i>Tree</i> . This is the first tree node to paste into the tree as a child of <code>pTre</code> .
<code>pTreEnd</code>	-	Pointer to a structure of type <i>Tree</i> . This is the last tree node to paste into the tree as a child of <code>pTre</code> .

## Return Value

No return value

## Notes

`pTreBeg` must have a set of successors one of which is `pTreEnd`.

[If `pTreBeg` does not equal `pTreEnd` then `pTreBeg` must precede `pTreEnd`.]

[`pTreBeg` must be the root node.]

## TrePasteRangeBeforeSibling

### Notes (cont)

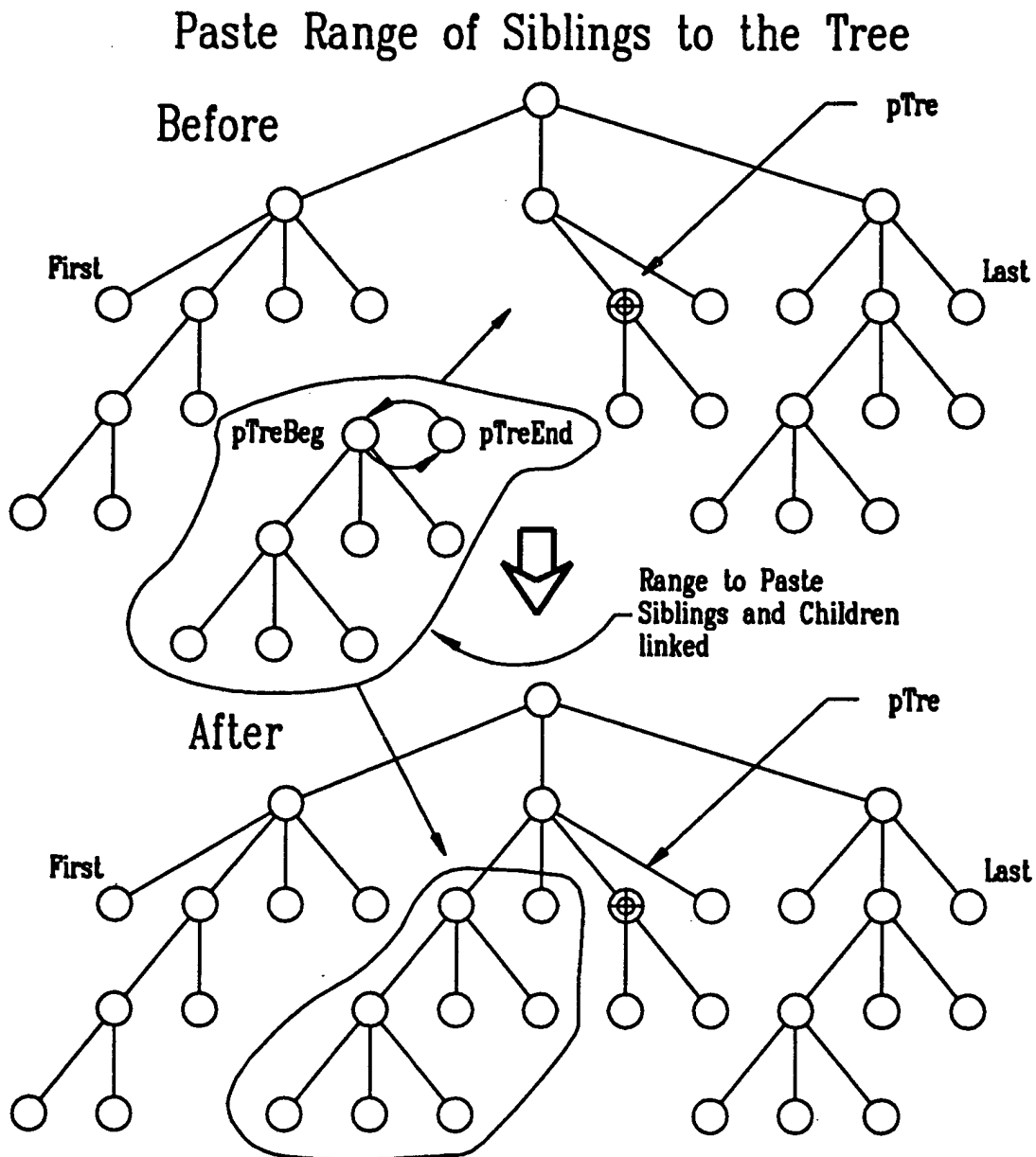
[If pTreBeg does not equal pTreEnd then pTreBeg must precede pTreEnd.]

[pTreBeg must be the root node.]

### See Also

TreCutRange, TreCutChildren, TrePasteRangeLastChild,  
TrePasteRangeFirstChild, TrePasteRangeAfterSibling

### Diagram



# TrePasteRangeBeforeSibling

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TrePasteRangeBeforeSibling( pTre, pTreBeg, pTreEnd )
PTRE      pTre;
PTRE      pTreBeg;
PTRE      pTreEnd;
```

Public Function

A macro is available for this function

## Purpose

The TrePasteRangeBeforeSibling function links the range of *Tree* nodes pTreBeg through pTreEnd as children of the parent of the *Tree* pTre. The range is linked so that the nodes are the predecessor siblings of pTre.

The tree node range must have been previously cut from a tree.

A range of tree nodes is defined as a beginning tree node and an ending tree node. The beginning node can equal the ending node. See *Tree* class section on range definition for more details.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The inserted tree node(s) become previous sibling(s) of this node.
pTreBeg	-	Pointer to a structure of type <i>Tree</i> . This is the first tree node to paste into the tree as a previous sibling of pTre.
pTreEnd	-	Pointer to a structure of type <i>Tree</i> . This is the last tree node to paste into the tree as a previous sibling of pTre.

## Return Value

No return value

## Notes

pTreBeg must have a set of successors one of which is pTreEnd.

[pTre must be a parent.]

## TrePasteRangeAfterSibling

### Notes (cont)

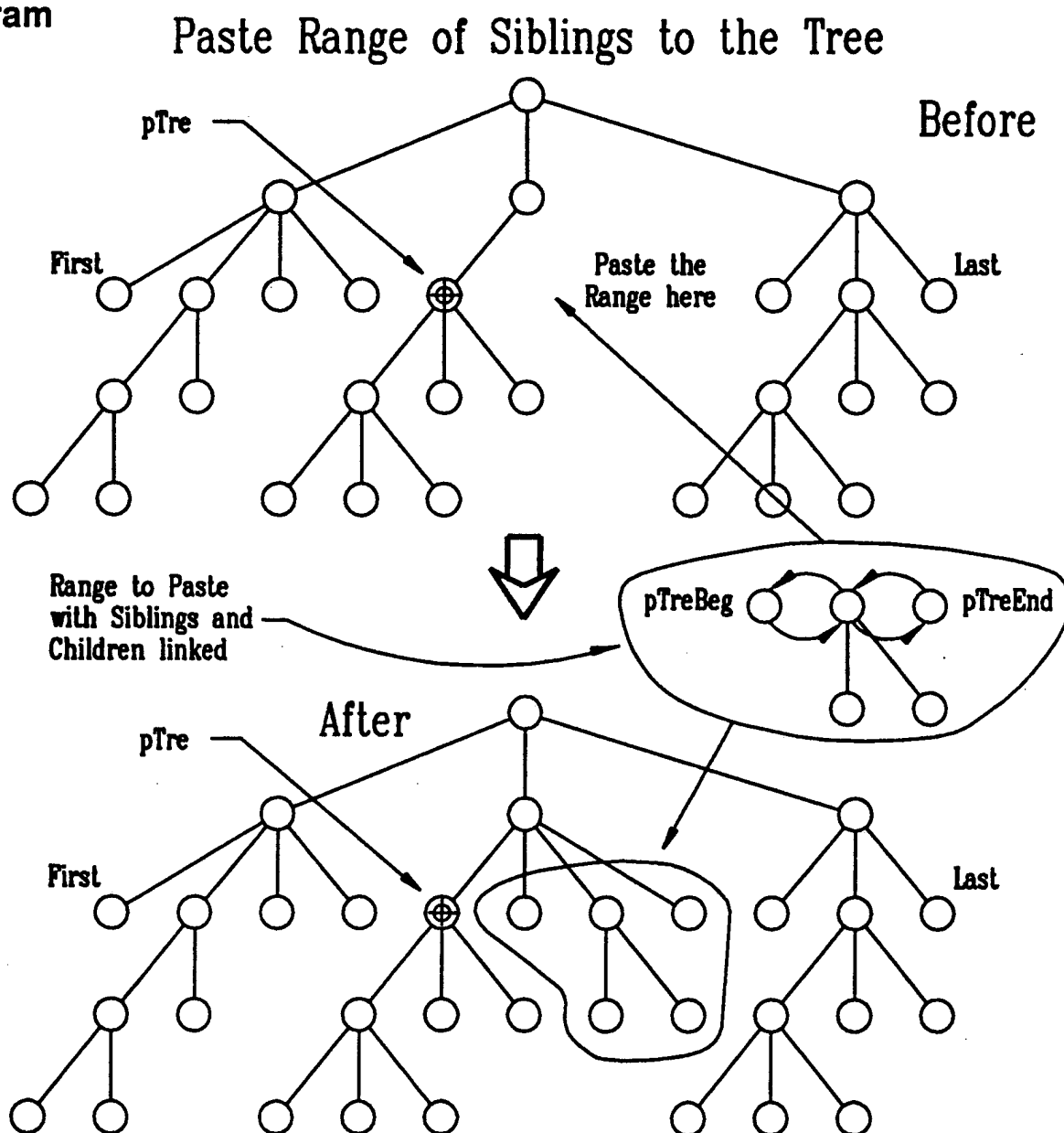
[If pTreBeg does not equal pTreEnd then pTreBeg must precede pTreEnd.]

[pTreBeg must be the root node.]

### See Also

TreCutRange, TreCutChildren, TrePasteRangeFirstChild,  
TrePasteRangeLastChild, TrePasteRangeBeforeSibling

### Diagram





# TrePasteRangeAfterSibling

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TrePasteRangeAfterSibling( pTre, pTreBeg, pTreEnd )
PTRE      pTre;
PTRE      pTreBeg;
PTRE      pTreEnd;
```

Public Function

A macro is available for this function

## Purpose

The `TrePasteRangeAfterSibling` function links the range of *Tree* nodes `pTreBeg` through `pTreEnd` as children of the parent of the *Tree* `pTre`. The range is linked so that the nodes are the successor siblings of `pTre`.

The tree node range must have been previously cut from a tree.

A range of tree nodes is defined as a beginning tree node and an ending tree node. The beginning node can equal the ending node. See *Tree* class section on range definition for more details.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The appended tree node(s) become next sibling(s) of this node.
<code>pTreBeg</code>	-	Pointer to a structure of type <i>Tree</i> . This is the first tree node to paste into the tree as a sibling of <code>pTre</code> .
<code>pTreEnd</code>	-	Pointer to a structure of type <i>Tree</i> . This is the last tree node to paste into the tree as a sibling of <code>pTre</code> .

## Return Value

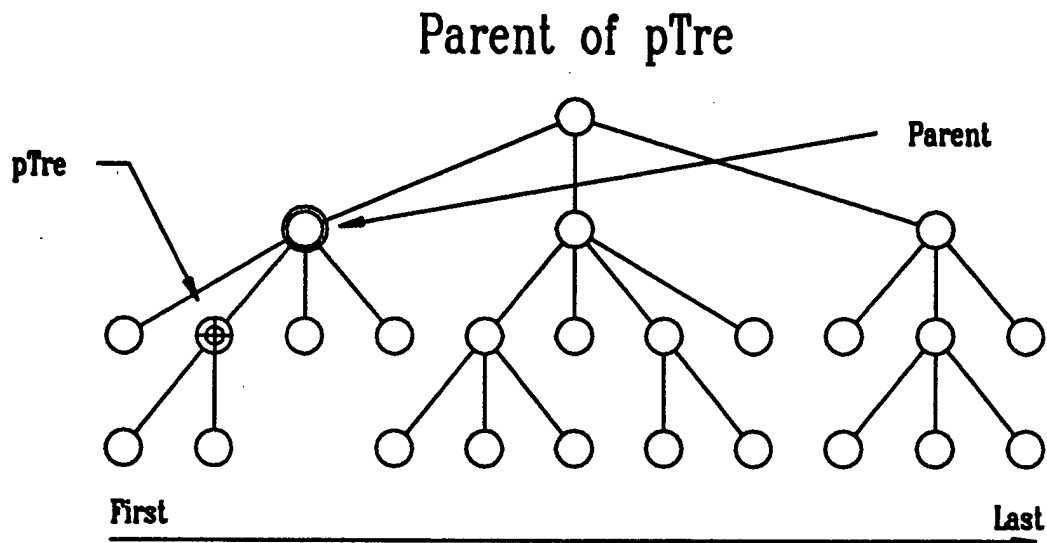
No return value

## Notes

`pTreBeg` must have a set of successors one of which is `pTreEnd`.

[`pTre` must be a parent.]

Diagram



# TreParent

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PTRE      TreParent( pTre )
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The TreParent function returns the parent of the *Tree* pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

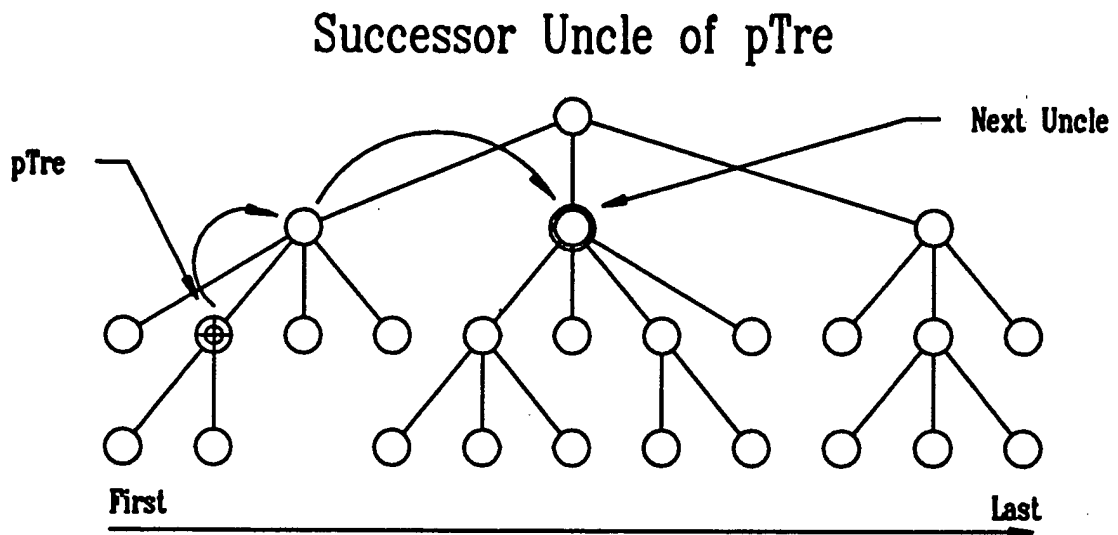
## Return Value

The return value from the TreParent function is a pointer to the parent of the *Tree* pTre. NULL is returned if no parent node exists.

## See Also

TreFirstChild, TreLastChild, TreLastLeaf, TreNext, TreNextUncle,  
TreNextPreOrder, TrePrev, TrePrevPreOrder

## Diagram



# TreNextUncle

---

## Summary

```
#include "cobjects.h"  
#include "tremac.h"
```

```
PTRE      TreNextUncle( pTre )  
PTRE      pTre;
```

Private Function

## Purpose

The TreNextUncle function returns the successor uncle of the *Tree* pTre. The next uncle is found by taking the successor of the parent of pTre. If the parent has no successor the function TreNextUncle is called recursively until the root node is reached or an uncle is found.

## Parameter - Description

---

pTre - Pointer to a structure of type *Tree*.

## Return Value

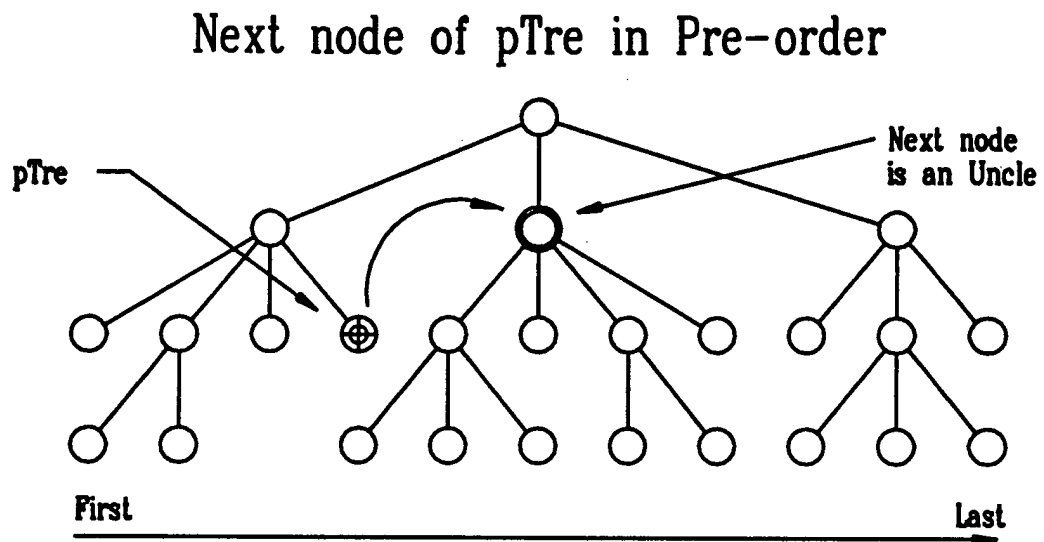
The return value from the TreNextUncle function is a pointer to the next uncle of the *Tree* pTre. NULL is returned if there is no uncle.

## See Also

TreFirstChild, TreLastChild, TreLastLeaf, TreNext, TreNextPreOrder, TreParent, TrePrev, TrePrevPreOrder

## TreNextPreOrder

### Diagram



# TreNextPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PTRE      TreNextPreOrder( pTre )
PTRE      pTre;
```

Private Function

## Purpose

The TreNextPreOrder function returns the pre-order successor of the *Tree* pTre. The pre-order successor is: i) the first child of pTre; ii) the successor sibling of pTre; ii) the uncle of pTre.

## Parameter - Description

---

pTre - Pointer to a structure of type *Tree*.

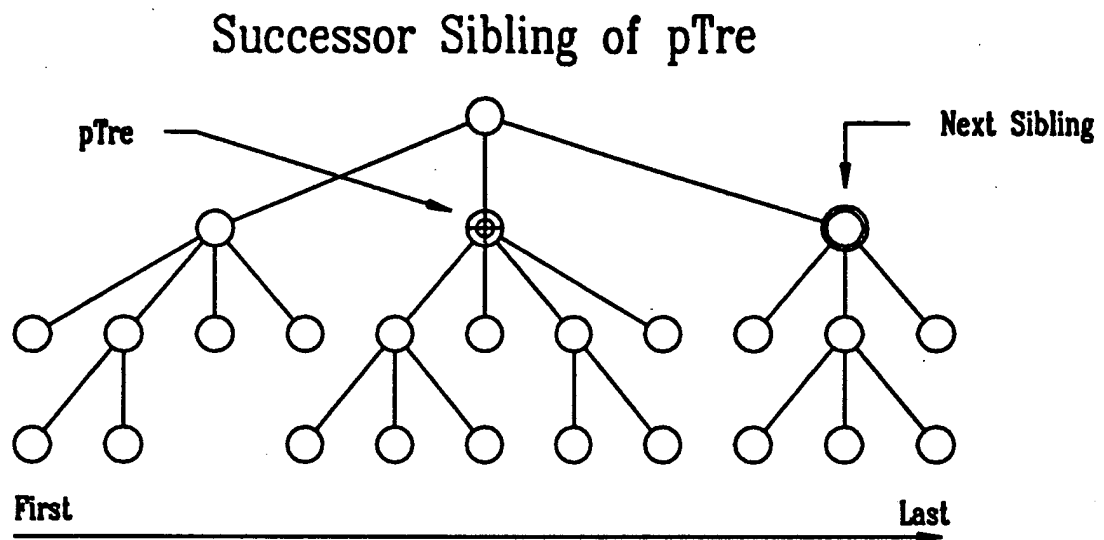
## Return Value

The return value from the TreNextPreOrder function is a pointer to the pre-order successor of the *Tree* pTre. NULL is returned if there is no successor.

## See Also

TreFirstChild, TreLastChild, TreLastLeaf, TreNext, TreNextUncle, TreParent, TrePrev, TrePrevPreOrder

Diagram





# TreNext

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PTRE      TreNext( pTre )
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The TreNext function returns the successor sibling of the *Tree* pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

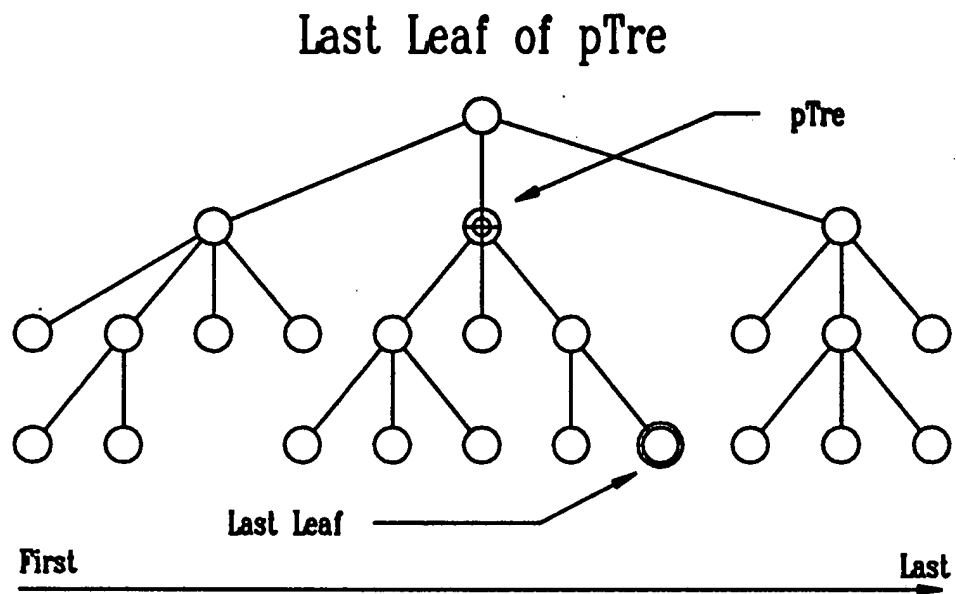
## Return Value

The return value from the TreNext function is a pointer to the successor sibling node of the *Tree* pTre. NULL is returned if there is no sibling.

## See Also

TreFirstChild, TreLastChild, TreLastLeaf, TreNextUncle, TreNextPreOrder, TreParent, TrePrev, TrePrevPreOrder

## Diagram



# TreLastLeaf

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PTRE      TreLastLeaf( pTre )
PTRE      pTre;
```

Private Function

## Purpose

The TreLastLeaf function returns the last leaf of the *Tree* pTre. The last leaf is found by calling TreLastLeaf recursively for the last child of pTre until a node is visited that has no children.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

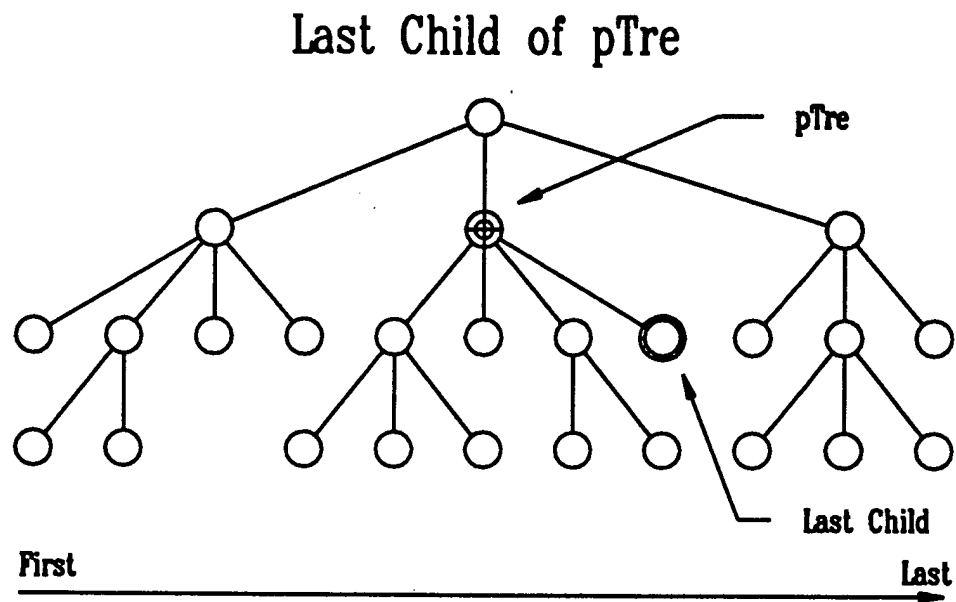
The return value from the TreLastLeaf function is a pointer to the last leaf of the *Tree* pTre. NULL is returned if there are no children.

## See Also

TreFirstChild, TreLastChild, TreNext, TreNextUncle, TreNextPreOrder, TreParent, TrePrev, TrePrevPreOrder

## TreLastChild

Diagram



# TreLastChild

---

## Summary

```
#include "cobjects.h"  
#include "tremac.h"
```

```
PTRE      TreLastChild( pTre )  
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The TreFirstChild function returns the last child of the *Tree* pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TreLastChild function is a pointer to the last child of the *Tree* pTre. NULL is returned if there are no children.

## See Also

TreFirstChild, TreLastLeaf, TreNext, TreNextUncle, TreNextPreOrder,  
TreParent, TrePrev, TrePrevPreOrder

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Bool      TrelsRoot( pTre )
PTRE      pTre;
```

Public Function

A macro is available for this function

## Purpose

The TrelsRoot function determines if the *Tree* pTre has no parent.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TrelsRoot function is True if the *Tree* pTre has no parent node, otherwise False is returned.

## See Also

TreHasChildren, TreHasSiblings, TreIsChild, TreIsDirectAncestor

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TrelsRoot function.

# TreIsDirectAncestor

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Bool      TreIsDirectAncestor( pTre, pTreA )
PTRE      pTre;
PTRE      pTreA;
```

Public Function

## Purpose

The TreIsDirectAncestor function determines if the *Tree* pTre is a parent, grand parent, or any other direct ancestor of pTre.

Parameter	-	Description
-----------	---	-------------

pTre	-	Pointer to a structure of type <i>Tree</i> .
pTreA	-	Pointer to a structure of type <i>Tree</i> . This tree node should be direct ancestor of pTre.

## Return Value

The return value from the TreIsDirectAncestor function is True if the *Tree* pTre is a parent, grand parent, or any other direct ancestor of pTre.

## See Also

TreHasChildren, TreHasSiblings, TreIsChild, TreIsRoot

## Example

Please refer to class test procedure TSTTRE<sub>1</sub>C for an example of the use of the TreIsDirectAncestor function.

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Bool      TrelChild( pTre )
PTRE      pTre;
```

Public Function

A macro is available for this function

## Purpose

The TrelChild function determines if the *Tree* pTre has a parent node.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TrelChild function is True if pTre has a parent, otherwise False is returned.

## See Also

TreHasChildren, TreHasSiblings, TrelRoot, TrelDirectAncestor

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TrelChild function.



# TreInit

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreInit( pTre )
PTRE      pTre;
```

Public Function

## Purpose

The TreInit function initializes the *Tree* object. The TreInit function should be the first function called when using the *Tree* class.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

No return value

## Notes

The last function to call when using the *Tree* class is TreDeInit.

## See Also

TreDeInit, TreDestroy

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreInit function.

# TreHasSiblings

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Bool      TreHasSiblings( pTre )
PTRE      pTre;
```

Public Function

A macro is available for this function

## Purpose

The TreHasSiblings function determines if the *Tree* pTre has any siblings.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TreHasSiblings function is True if the *Tree* pTre has any siblings, otherwise False is returned.

## See Also

TreHasChildren, TreIsChild, TreIsRoot, TreIsDirectAncestor

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreHasSiblings function.

# TreHasChildren

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Bool      TreHasChildren( pTre )
PTRE      pTre;
```

Public Function

A macro is available for this function

## Purpose

The TreHasChildren function determines if the *Tree* pTre has any children.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TreHasChildren function is True if the *Tree* pTre has any children, otherwise False is returned.

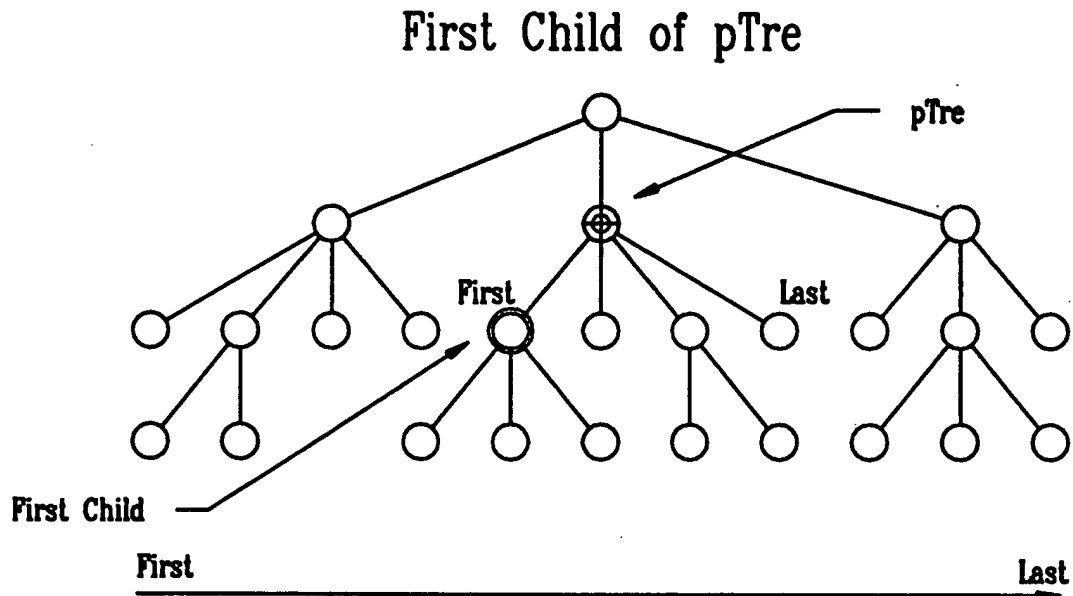
## See Also

TreHasSiblings, TreIsChild, TreIsRoot, TreIsDirectAncestor

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreHasChildren function.

## Diagram



# TreFirstChild

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PTRE      TreFirstChild( pTre )
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The TreFirstChild function returns the first child of the *Tree* pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TreFirstChild function is a pointer to the first child of the *Tree* pTre. NULL is returned if there are no children.

## See Also

TreLastChild, TreLastLeaf, TreNext, TreNextUncle, TreNextPreOrder, TreParent, TrePrev, TrePrevPreOrder

# TreDestroyChildren

---

## Summary

```
#include "cobjects.h"  
#include "tremac.h"
```

```
Void      TreDestroyChildren( pTre )  
PTRE      pTre;
```

Public Function

## Purpose

The TreDestroyChildren function sends a message to the clients of any children of the *Tree* pTre to destroy themselves.

## Parameter - Description

---

pTre - Pointer to a structure of type *Tree*.

## Return Value

No return value

## See Also

TreClearChildren

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreDestroyChildren function.

# TreDestroy

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreDestroy( pTre )
PTRE      pTre;
```

Public Function

## Purpose

The TreDestroy function deallocates the memory used by the object and deinitializes the *Tree* object. The *Tree* pTre should not be referenced after this function call since its memory will have been deallocated.

The *Tree* pTre and any children of pTre will be cleared prior to deinitializing pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

No return value

## Notes

[pTre must not have a sub-object.]

## See Also

TreDeInit, TreDestroyChildren, TreInit

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreDestroy function.

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreDeInit( pTre )
PTRE      pTre;
```

Public Function

## Purpose

The TreDeInit function deinitializes the *Tree* object. The TreDeInit function should be the last function called when using the *Tree* class.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

No return value

## Notes

The first function to call when using the *Tree* class is TreInit.

[pTre must be the root node.]

[pTre must have no child nodes.]

## See Also

TreDestroy, TreDestroyChildren, TreInit

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreDeInit function.

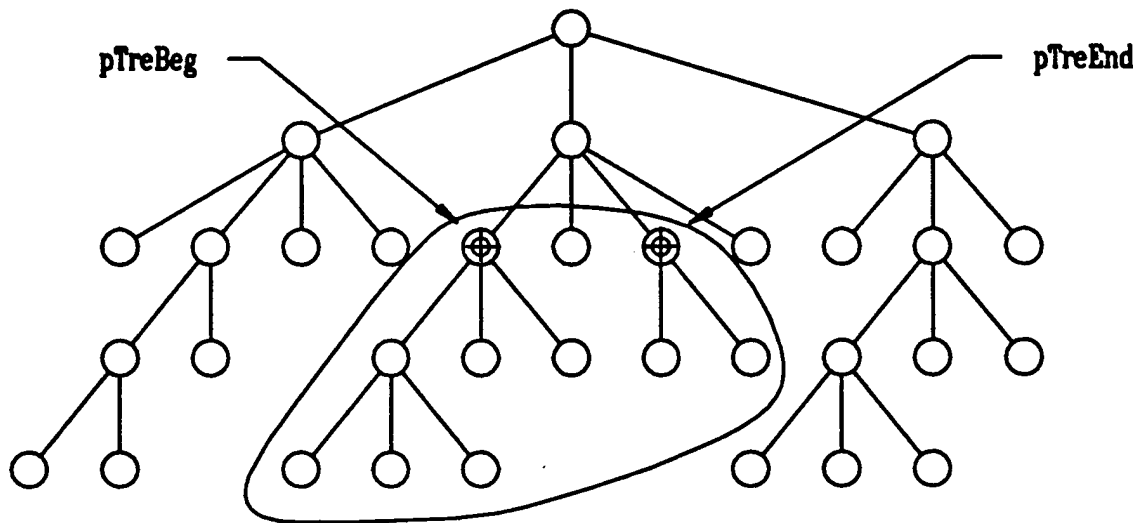


# TreCutRange

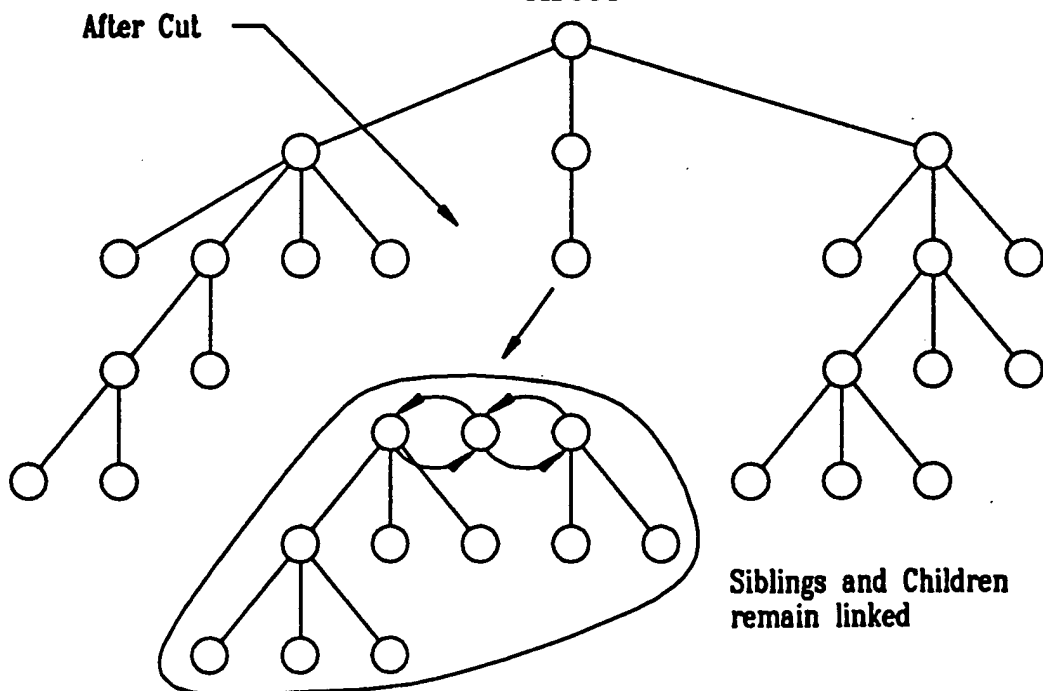
## Diagram

Cut a Range from the Tree

Before



After



# TreCutRange

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreCutRange( pTreBeg, pTreEnd )
PTRE      pTreBeg;
PTRE      pTreEnd;
```

Public Function

A macro is available for this function

## Purpose

The TreCutRange function unlinks a range of *Tree* nodes pTreBeg through pTreEnd from the *Tree*. The tree node preceding pTreBeg will be linked to the successor tree node of pTreEnd. The range pTreBeg through pTreEnd become roots but the sibling connections remain.

A range of tree nodes is defined as a beginning tree node and an ending tree node. The beginning node can equal the ending node. See *Tree* class section on range definition for more details.

Parameter	-	Description
-----------	---	-------------

---

pTreBeg	-	Pointer to a structure of type <i>Tree</i> . This is the first tree node to cut from the tree.
pTreEnd	-	Pointer to a structure of type <i>Tree</i> . This is the last tree node to cut from the tree.

## Return Value

No return value

## Notes

[If pTreBeg does not equal pTreEnd then pTreBeg must precede pTreEnd.]

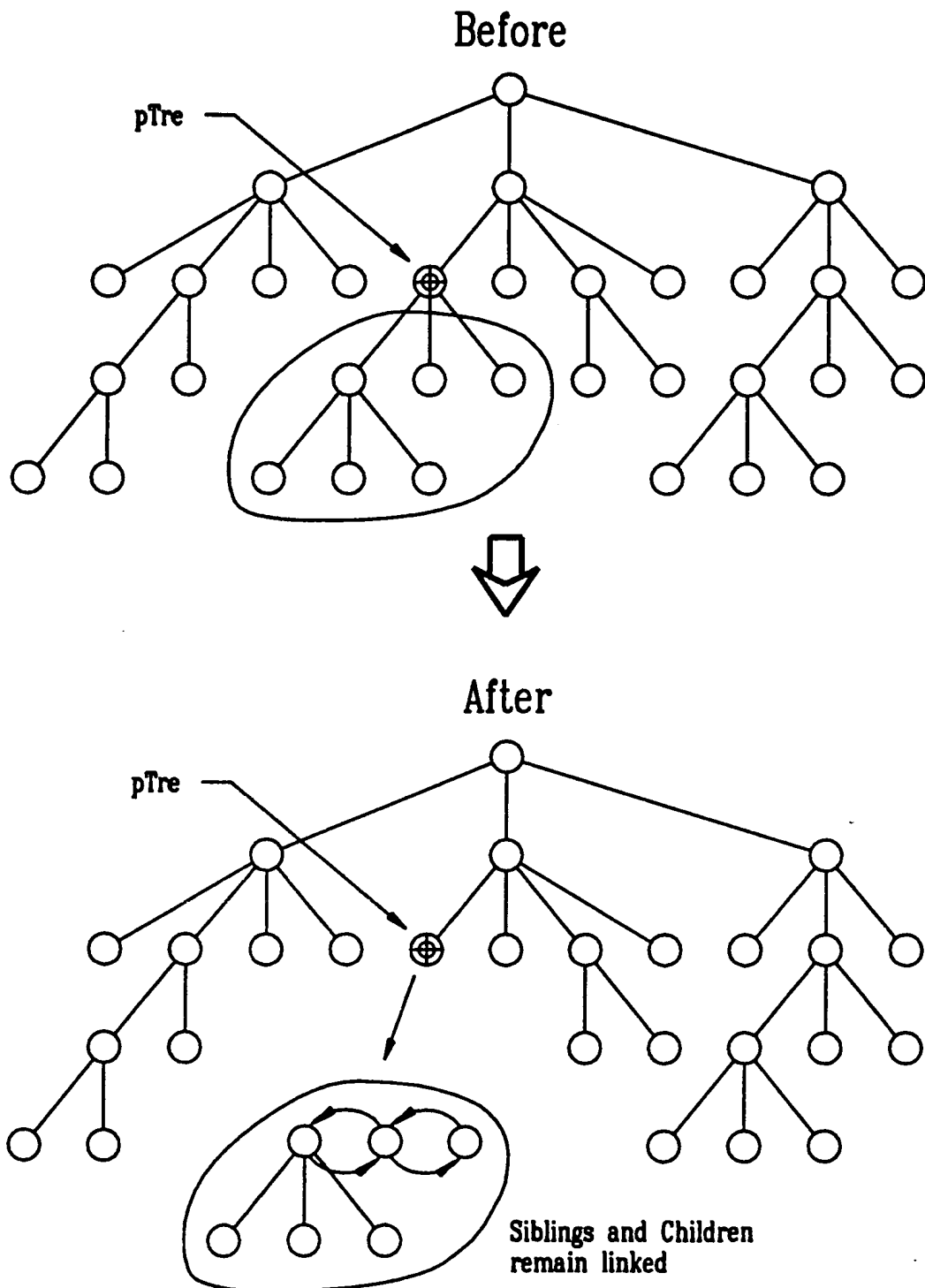
## See Also

TreCutChildren, TrePasteRangeFirstChild, TrePasteRangeLastChild,  
TrePasteRangeAfterSibling, TrePasteRangeBeforeSibling

# TreCutChildren

## Diagram

Cut all Children from the Tree



# TreCutChildren

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreCutChildren( pTre )
PTRE      pTre;
```

Public Function

A macro is available for this function

## Purpose

The TreCutChildren function unlinks any children of the *Tree* pTre from the *Tree*.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

No return value

## See Also

TreCutRange, TrePasteRangeFirstChild, TrePasteRangeLastChild,  
TrePasteRangeAfterSibling, TrePasteRangeBeforeSibling

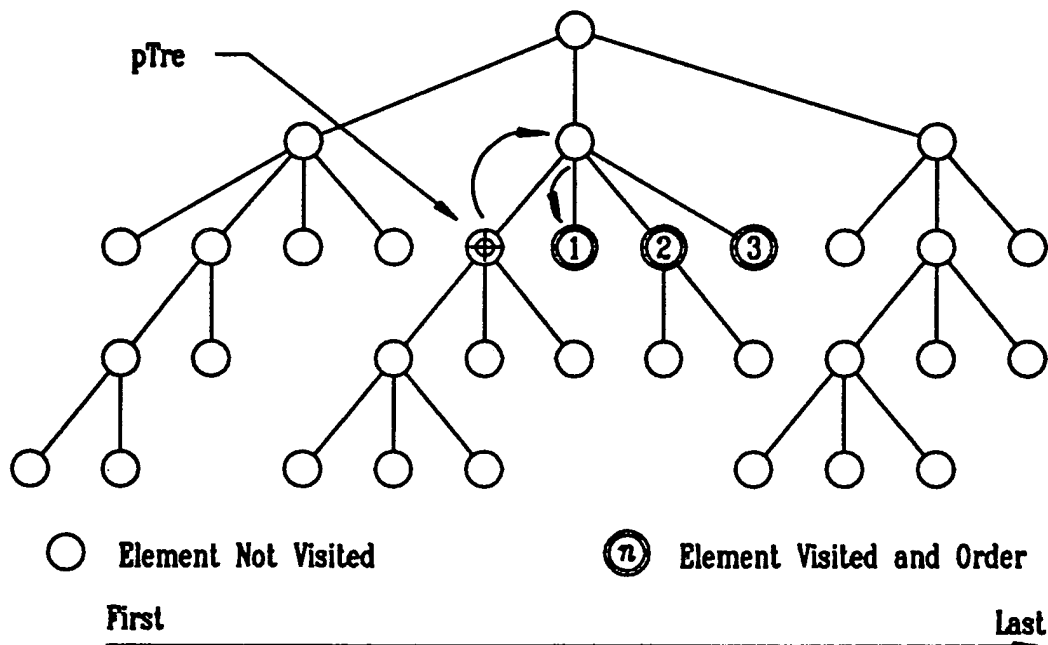
# TreClientVisitSuccessors

## See Als

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitDescInOrderBwd,  
TreClientVisitLeaves, TreClientVisitParents, TreClientVisitRange

## Diagram

### Walk the Tree for Successors



# TreClientVisitSuccessors

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitSuccessors( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitSuccessors function walks the *Tree* and calls a *Tree* client function for each tree node visited. The *Tree* is walked in a forward direction for all successors and starts with the tree node that is the next sibling to pTre.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

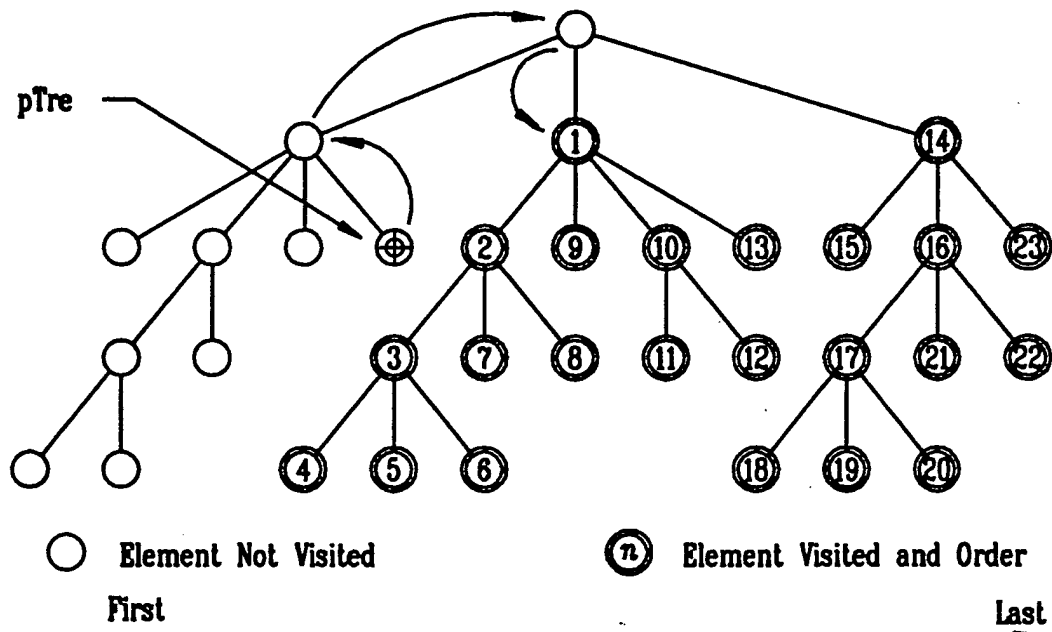
# TreClientVisitSuccPreOrder

## See Als

TreClientVisitBranchInOrder, TreClientVisitPreOrder, TreClientVisitChildren,  
TreClientVisitChildrenBwd, TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitDescInOrderBwd,  
TreClientVisitLeaves, TreClientVisitParents, TreClientVisitRange,  
TreClientVisitSuccessors

## Diagram

Walk the Tree, all successors in Pre-order



# TreClientVisitSuccPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitSuccPreOrder( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitSuccPreOrder function walks the *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked starting with the pre-order successor to pTre and visits all pre-order successors of pTre.

The *Block* pBlk contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The PreOrder successor to this node is the first visited.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.



# TreClientVisitRange

## Notes

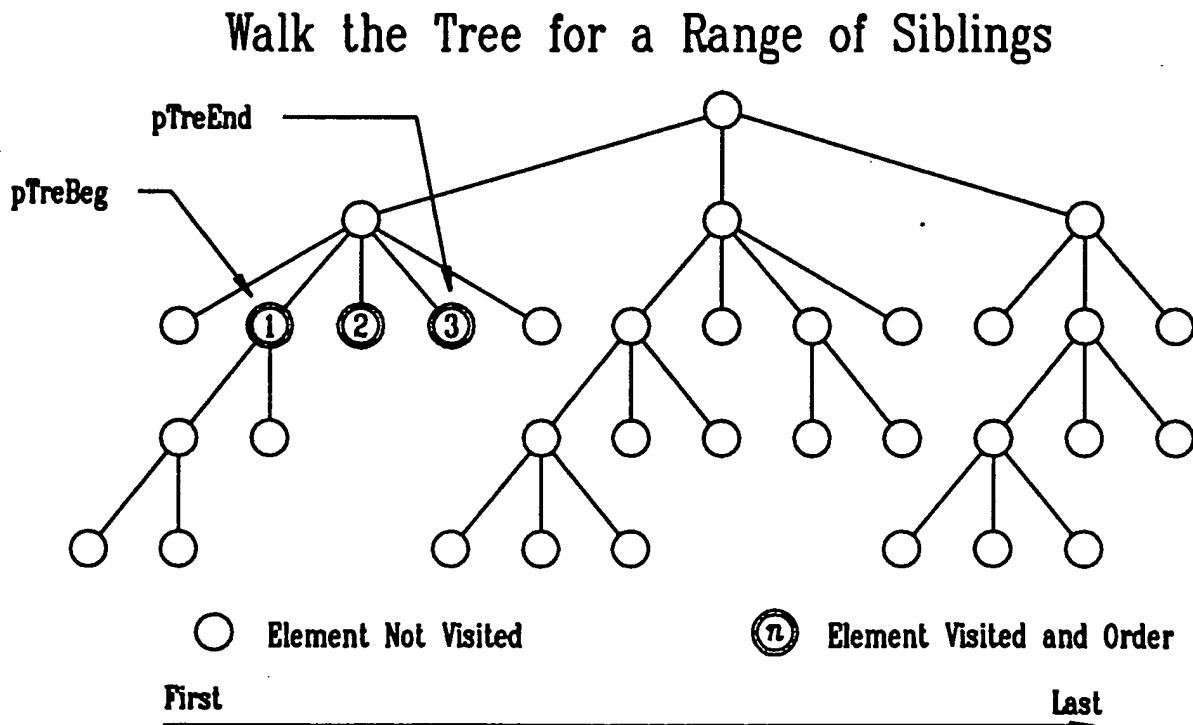
The client function may return a value but it is ignored.

[If pTreBeg does not equal pTreEnd then pTreBeg must precede pTreEnd.]

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitDescInOrderBwd,  
TreClientVisitLeaves, TreClientVisitParents, TreClientVisitSuccessors

## Diagram



# TreClientVisitRange

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
MediumInt    TreClientVisitRange( pTreBeg, pTreEnd, offset, pBlk )
PTRE         pTreBeg;
PTRE         pTreEnd;
MediumInt    offset;
PBLK         pBlk;
```

Public Function

## Purpose

The TreClientVisitRange function walks the *Tree* and calls a *Tree* client function for each tree node visited. The *Tree* is walked for a range of sibling nodes, pTreBeg through pTreEnd. The siblings are visited in a forward direction.

The *Block* pBlk contains the client function and an optional list of arguments.

A range of tree nodes is defined as a beginning tree node and an ending tree node. The beginning node can equal the ending node. See *Tree* class section on range definition for more details.

Parameter	-	Description
-----------	---	-------------

---

pTreBeg	-	Pointer to a structure of type <i>Tree</i> . This is the first tree node in a range.
pTreEnd	-	Pointer to a structure of type <i>Tree</i> . This is the last tree node in a range.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

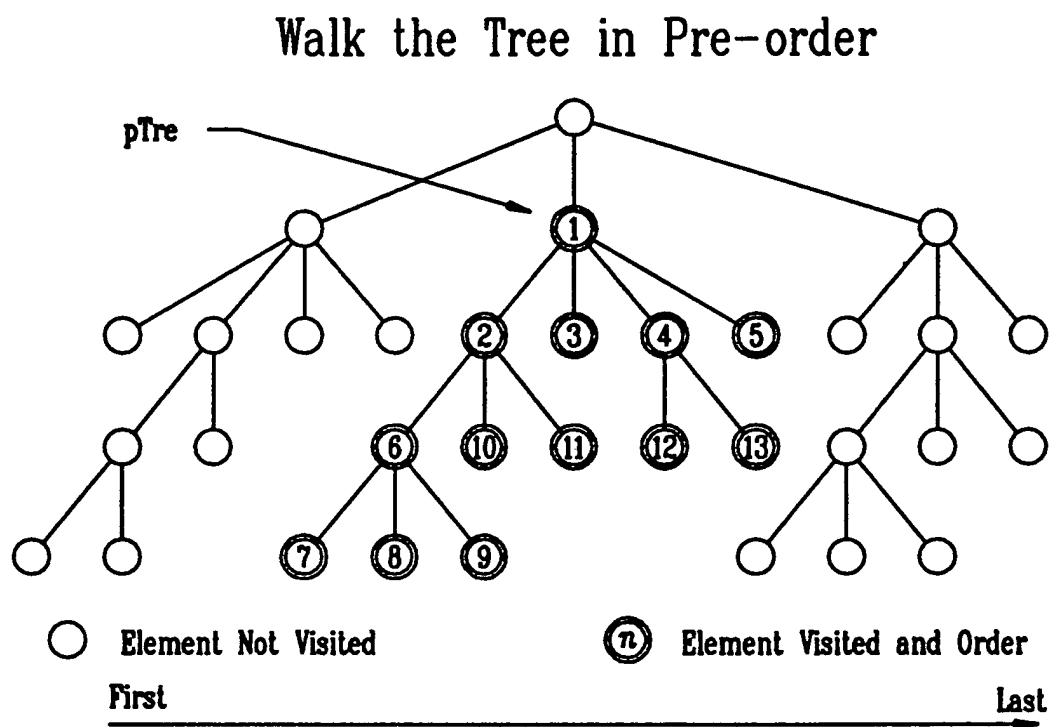
The return value from the TreClientVisitRange function is the number of times tree nodes are visited.

# TreClientVisitPreOrder

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitChildren, TreClientVisitChildrenBwd, TreClientVisitInOrder,  
TreClientVisitInOrderBwd, TreClientVisitDescBranchInOrder,  
TreClientVisitDescPreOrder, TreClientVisitDescInOrder,  
TreClientVisitDescInOrderBwd, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

## Diagram



# TreClientVisitPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitPreOrder( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitPreOrder function walks a *Tree* and calls the *Tree* client function for each tree node visited. The walk visits pTre first and calls TreClientVisitPreOrder recursively for each of its children.

The *Block* pBlk contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

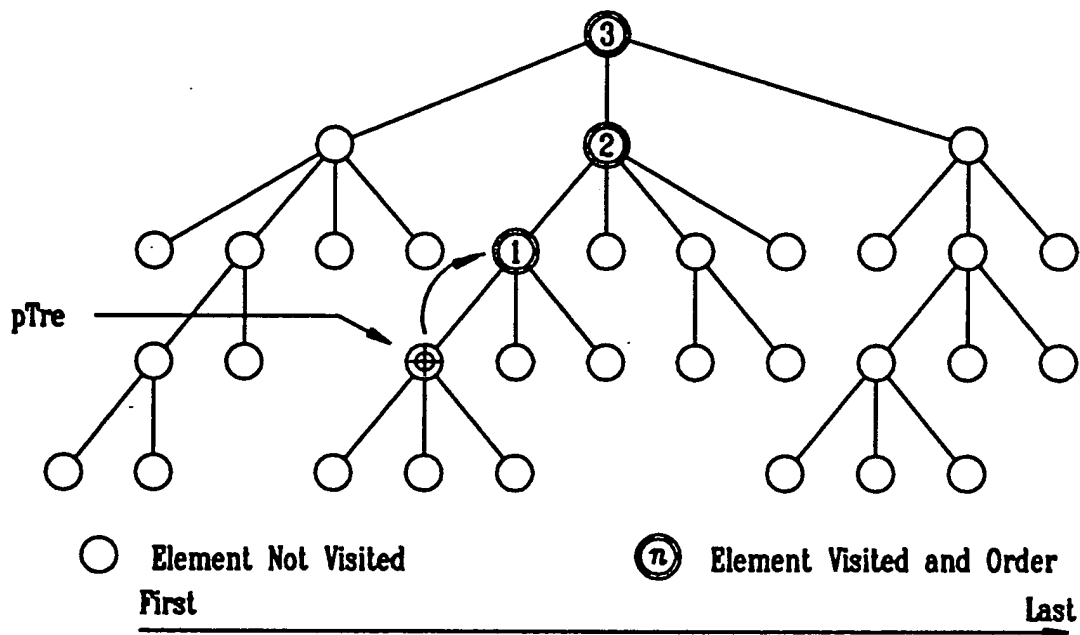
# TreClientVisitParents

## See Als

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitDescInOrderBwd,  
TreClientVisitLeaves, TreClientVisitRange, TreClientVisitSuccessors

## Diagram

Walk the Tree Nearest Parents First



# TreClientVisitParents

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitParents( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitParents function walks a *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked by calling TreClientVisitParents recursively for the parent of pTre.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

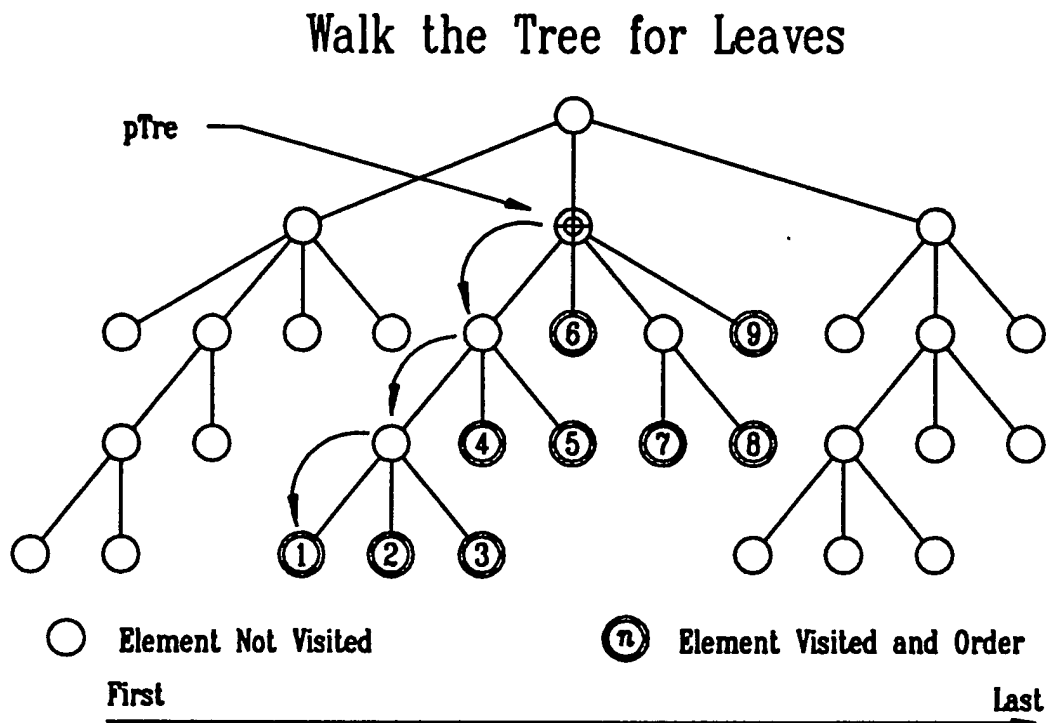
The client function may return a value but it is ignored.

# TreClientVisitLeaves

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitDescInOrderBwd,  
TreClientVisitParents, TreClientVisitRange, TreClientVisitSuccessors

## Diagram



# TreClientVisitLeaves

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitLeaves( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitLeaves function walks the *Tree* and calls the *Tree* client function for each tree node visited. The function calls TreClientVisitLeaves recursively for each of its children. The nodes are visited if they are leaves (have no children).

The *Block* pBlk contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

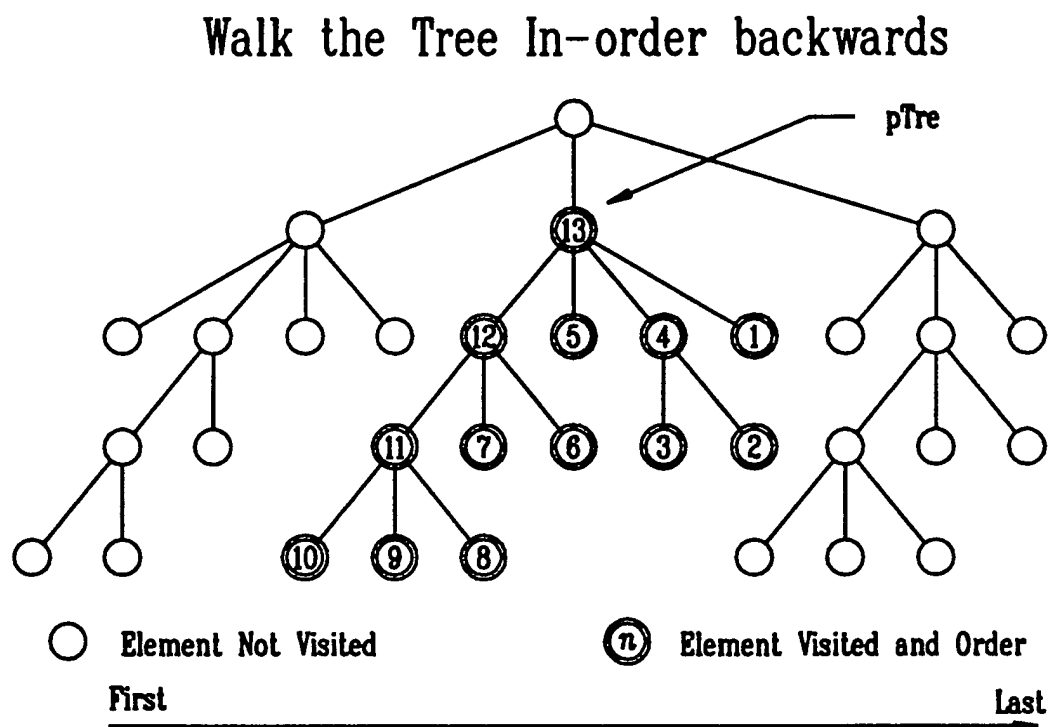


# TreClientVisitInOrderBwd

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitDescBranchInOrder,  
TreClientVisitDescPreOrder, TreClientVisitDescInOrder,  
TreClientVisitDescInOrderBwd, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

## Diagram



# TreClientVisitInOrderBwd

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitInOrderBwd( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitInOrderBwd function walks a *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked by calling TreClientVisitInOrderBwd recursively for the children of pTre, and then visiting pTre itself. The nodes are visited last to first.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

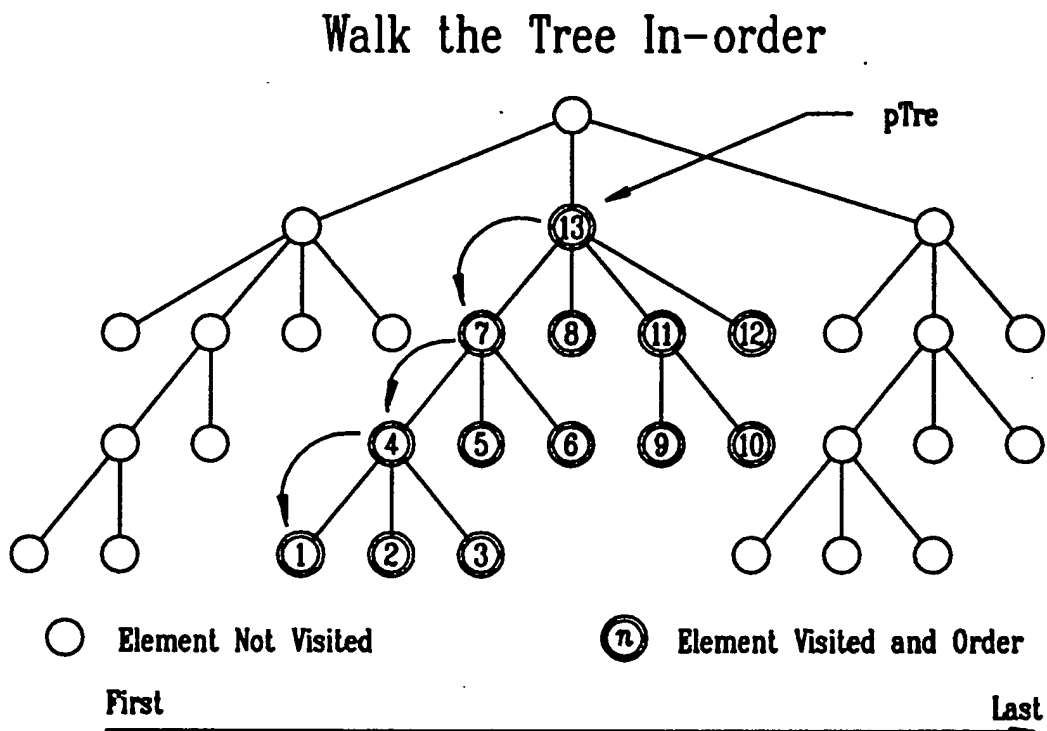
The client function may return a value but it is ignored.

# TreClientVisitInOrder

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrderBwd, TreClientVisitDescBranchInOrder,  
TreClientVisitDescPreOrder, TreClientVisitDescInOrder,  
TreClientVisitDescInOrderBwd, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

## Diagram



# TreClientVisitInOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitInOrder( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The `TreClientVisitInOrder` function walks a *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked by calling `TreClientVisitInOrder` recursively for the children of `pTre`, and then visiting `pTre` itself. The nodes are visited in a forward direction.

The *Block* `pBlk` contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
<code>offset</code>	-	The distance in bytes between the <i>Tree</i> <code>pTre</code> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

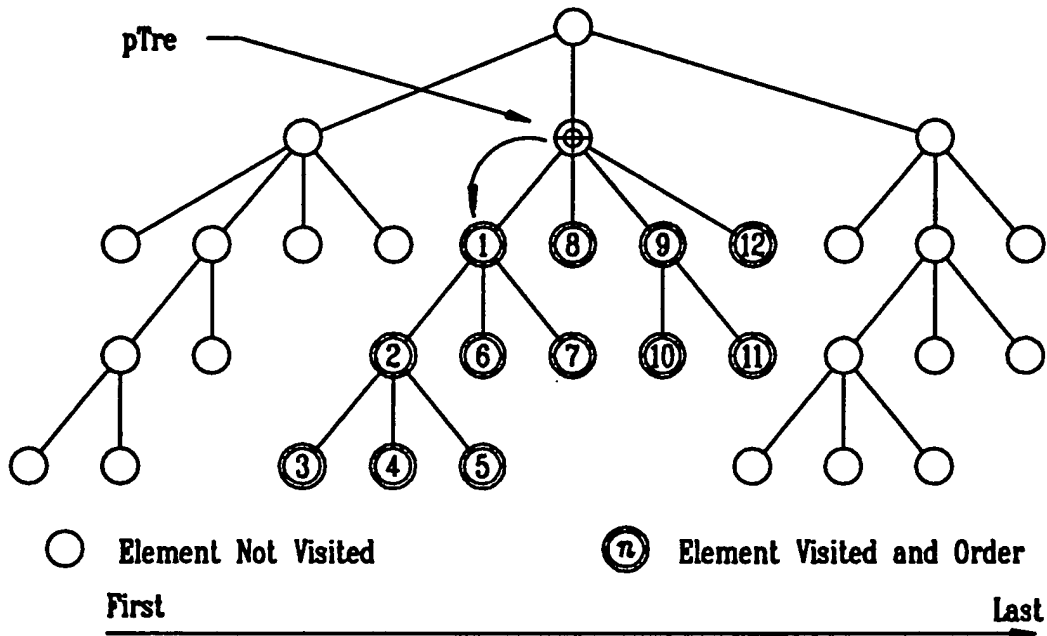
# TreClientVisitDescPreOrder

## See Als

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescInOrder,  
TreClientVisitDescInOrderBwd, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

## Diagram

Walk the Tree Descendents in Pre-order



# TreClientVisitDescPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitDescPreOrder( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitDescPreOrder function walks the *Tree* and calls a *Tree* client function for each tree node visited. The *Tree* is walked by calling TreClientVisitPreOrder for each of its children.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

# TreClientVisitDescLeaves

---

## See Als

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitDescInOrderBwd,  
TreClientVisitLeaves, TreClientVisitParents, TreClientVisitRange,  
TreClientVisitSuccessors

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreClientVisitDescLeaves function.

# TreClientVisitDescLeaves

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitDescLeaves( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Private Function

## Purpose

The TreClientVisitDescLeaves function walks the *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked by calling TreClientVisitLeaves for each of its children. The nodes are visited if they are leaves (have no children). The nodes are visited in a forward direction.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.



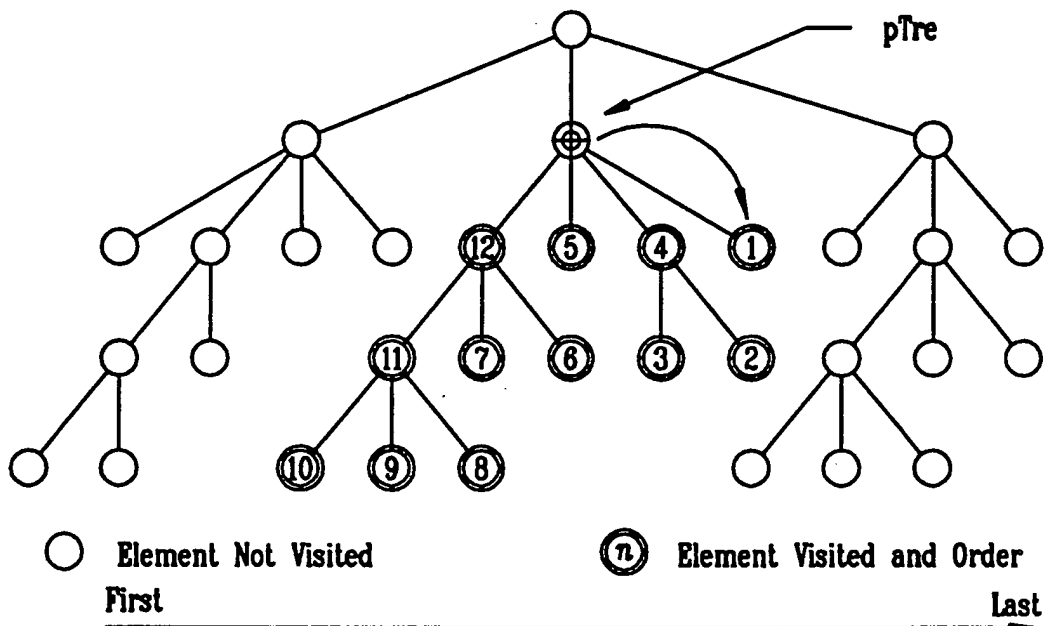
## TreClientVisitDescInOrderBwd

### See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

### Diagram

Walk the Tree Descendents In-order backwards



# TreClientVisitDescInOrderBwd

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitDescInOrderBwd( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitDescInOrderBwd function walks the *Tree* and calls a *Tree* client function for each tree node visited. The *Tree* is walked by calling TreClientVisitInOrderBwd for the children of pTre. The nodes are visited last to first.

The *Block* pBlk contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

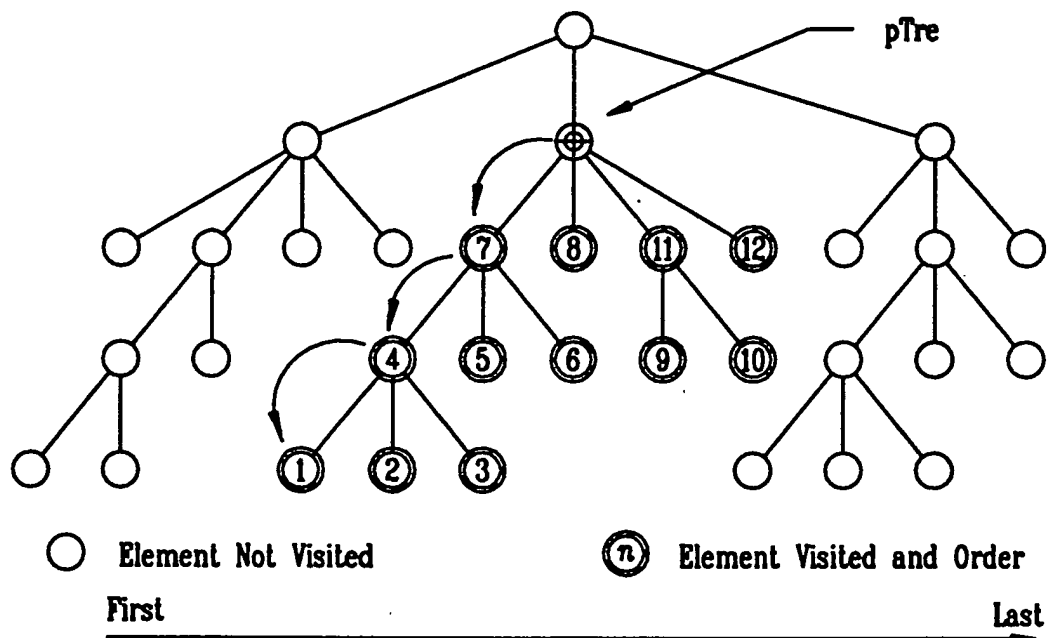
# TreClientVisitDescInOrder

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitChildrenBwd,  
TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrderBwd, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

## Diagram

### Walk the Tree Descendents In-order



# TreClientVisitDescInOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitDescInOrder( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The `TreClientVisitDescInOrder` function walks the *Tree* and calls a *Tree* client function for each tree node visited. The *Tree* is walked by calling `TreClientVisitInOrder` for the children of `pTre`. The nodes are visited in a forward direction.

The *Block* `pBlk` contains the client function and an optional list of arguments.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
<code>offset</code>	-	The distance in bytes between the <i>Tree</i> <code>pTre</code> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.



# TreClientVisitDescBranchInOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitDescBranchInOrder( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitDescBranchInOrder function walks a *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked by calling TreClientVisitBranchInOrder for each of its children. The nodes are visited if they are branches (have children).

The *Block* pBlk contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

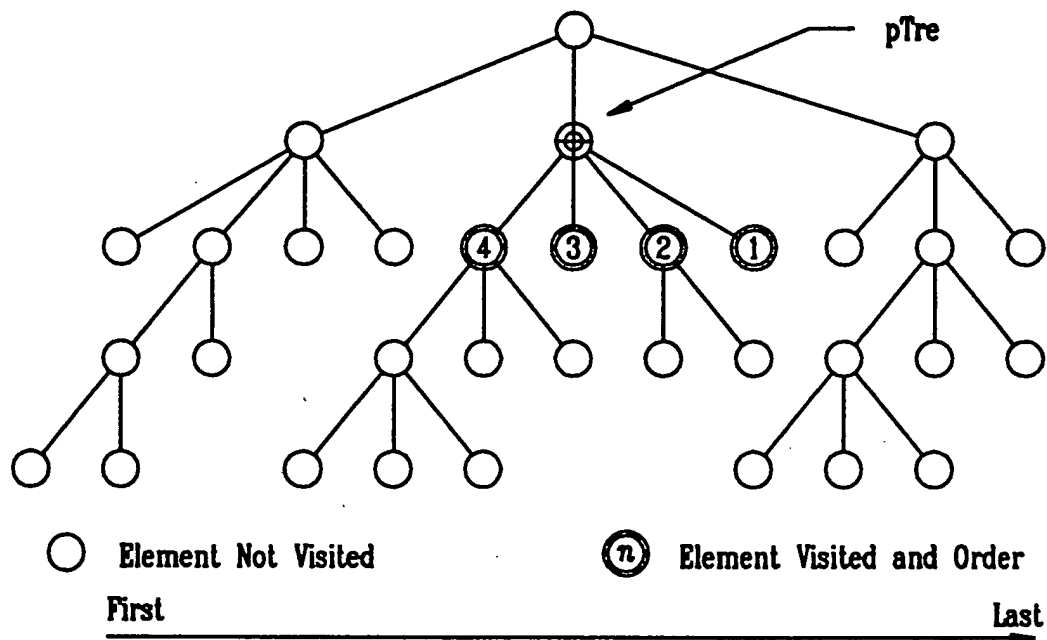
# TreClientVisitChildrenBwd

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildren, TreClientVisitInOrder,  
TreClientVisitInOrderBwd, TreClientVisitDescBranchInOrder,  
TreClientVisitDescPreOrder, TreClientVisitDescInOrder,  
TreClientVisitDescInOrderBwd, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

## Diagram

Walk the Tree for all Children Backwards



# TreClientVisitChildrenBwd

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitChildrenBwd( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitChildrenBwd function walks a *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked for each of the children of the *Tree* pTre starting with its last child and ending with its first child.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

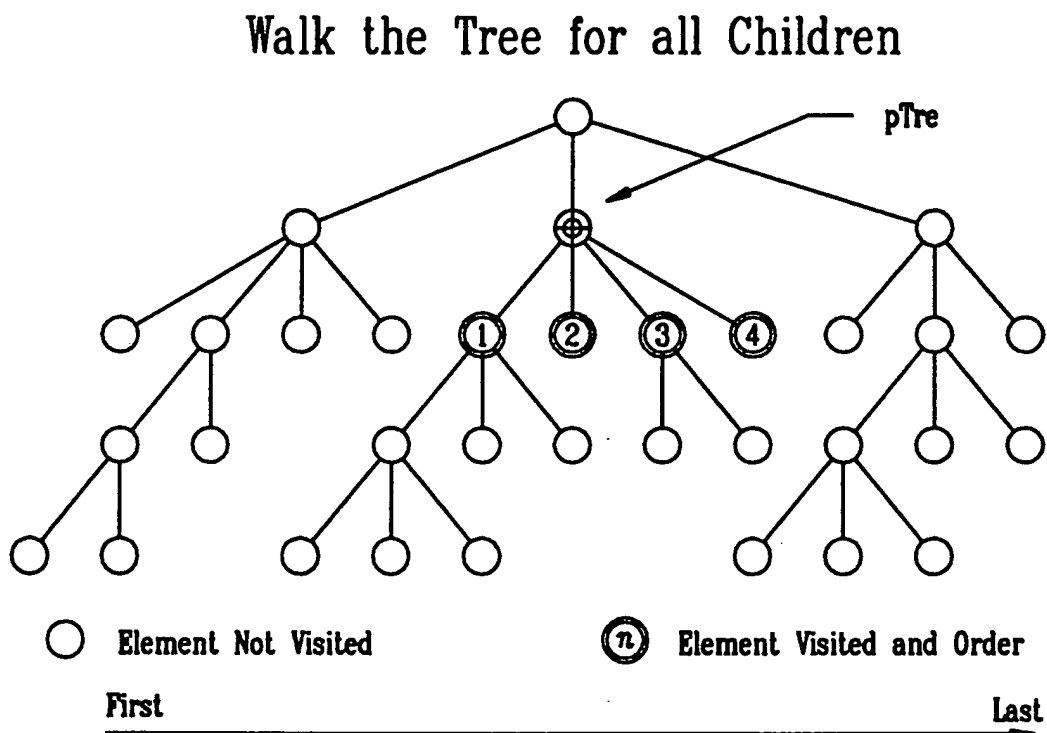


# TreClientVisitChildren

## See Also

TreClientVisitSuccPreOrder, TreClientVisitBranchInOrder,  
TreClientVisitPreOrder, TreClientVisitChildrenBwd, TreClientVisitInOrder,  
TreClientVisitInOrderBwd, TreClientVisitDescBranchInOrder,  
TreClientVisitDescPreOrder, TreClientVisitDescInOrder,  
TreClientVisitDescInOrderBwd, TreClientVisitLeaves, TreClientVisitParents,  
TreClientVisitRange, TreClientVisitSuccessors

## Diagram



# TreClientVisitChildren

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitChildren( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitChildren function walks a *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked for each of the children of the *Tree* pTre starting with its first child and ending with its last child.

The *Block* pBlk contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

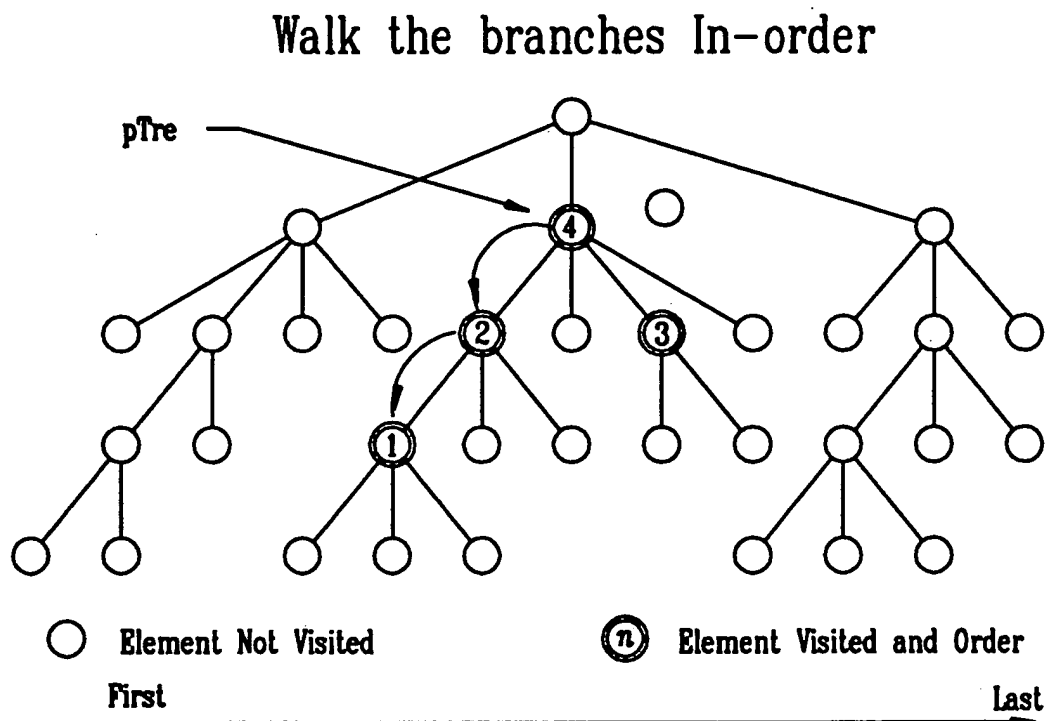
The client function may return a value but it is ignored.

# TreClientVisitBranchInOrder

## See Als

TreClientVisitSuccPreOrder, TreClientVisitPreOrder, TreClientVisitChildren,  
TreClientVisitChildrenBwd, TreClientVisitInOrder, TreClientVisitInOrderBwd,  
TreClientVisitDescBranchInOrder, TreClientVisitDescPreOrder,  
TreClientVisitDescInOrder, TreClientVisitDescInOrderBwd,  
TreClientVisitLeaves, TreClientVisitParents, TreClientVisitRange,  
TreClientVisitSuccessors

## Diagram



# TreClientVisitBranchInOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClientVisitBranchInOrder( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientVisitBranchInOrder function walks a *Tree* and calls the *Tree* client function for each tree node visited. The tree is walked by calling TreClientVisitBranchInOrder recursively for each of its children, and then visiting pTre itself. The nodes are visited if they are branches (have children).

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

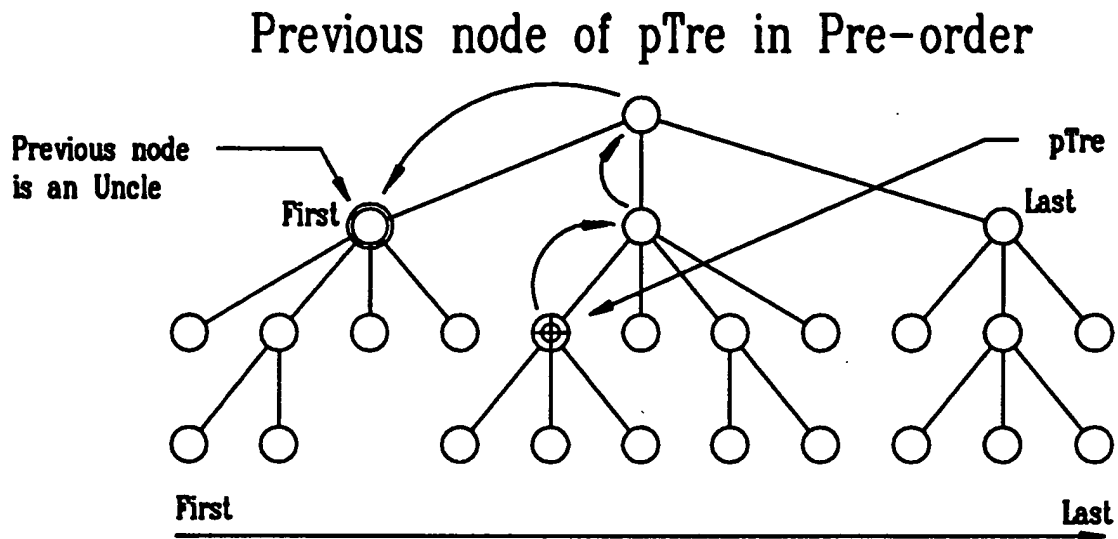
No return value

## Notes

The client function may return a value but it is ignored.

# TreClientPrevPreOrder

## Diagram



# TreClientPrevPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientPrevPreOrder( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientPrevPreOrder function returns the client of the pre-order predecessor the *Tree* pTre. The pre-order predecessor is the previous sibling of pTre. If pTre has no previous sibling the function TreClientPrevPreOrder is called recursively for the parent of pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

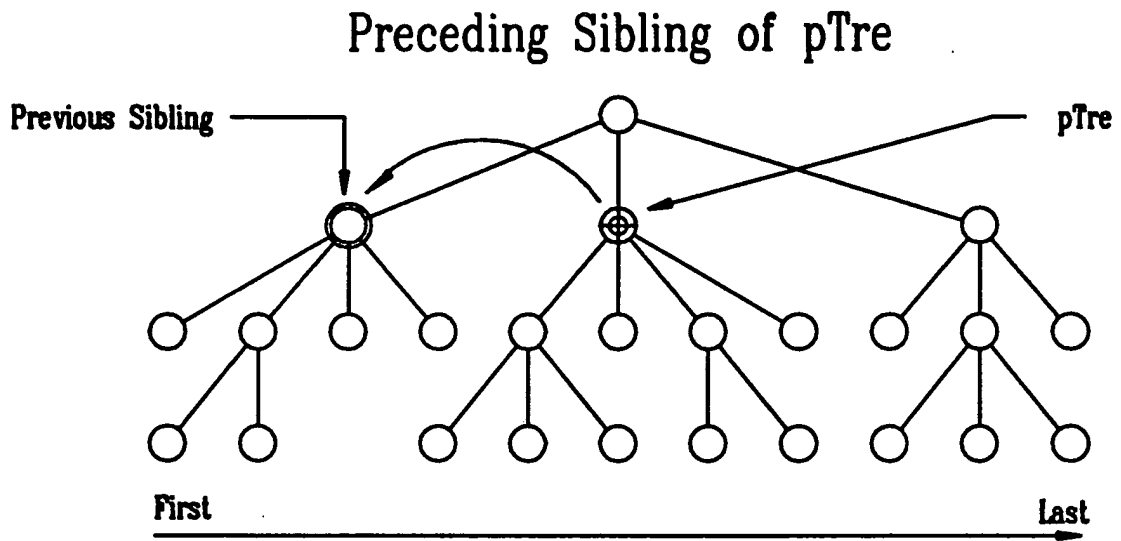
The return value from the TreClientPrevPreOrder function is a pointer to the client of the pre-order predecessor of the *Tree* pTre. NULL is returned if pTre has no parent.

## See Also

TreClient, TreClientFirstChild, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextPreOrder, TreClientNextUncle, TreClientParent, TreClientPrev

# TreClientPrev

## Diagram



## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientPrev( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientPrev function returns the client of the preceding sibling of the *Tree* pTre.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the TreClientPrev function is a pointer to the client of the previous sibling of the *Tree* pTre. NULL is returned if no previous sibling exists.

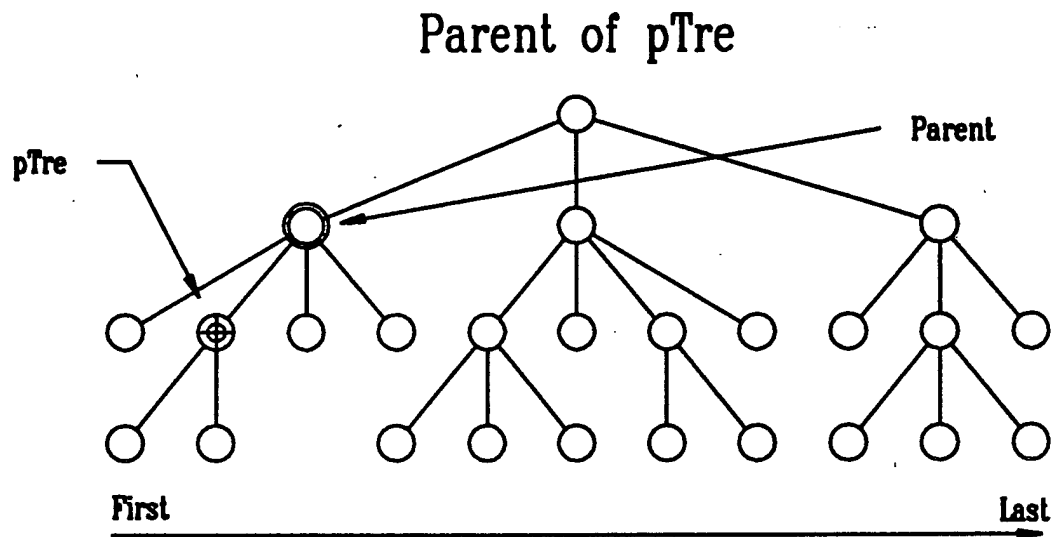
## See Also

TreClient, TreClientFirstChild, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextPreOrder, TreClientNextUncle, TreClientParent, TreClientPrevPreOrder



# TreClientParent

## Diagram



# TreClientParent

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientParent( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientParent function returns the client of the parent of the *Tree* pTre.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

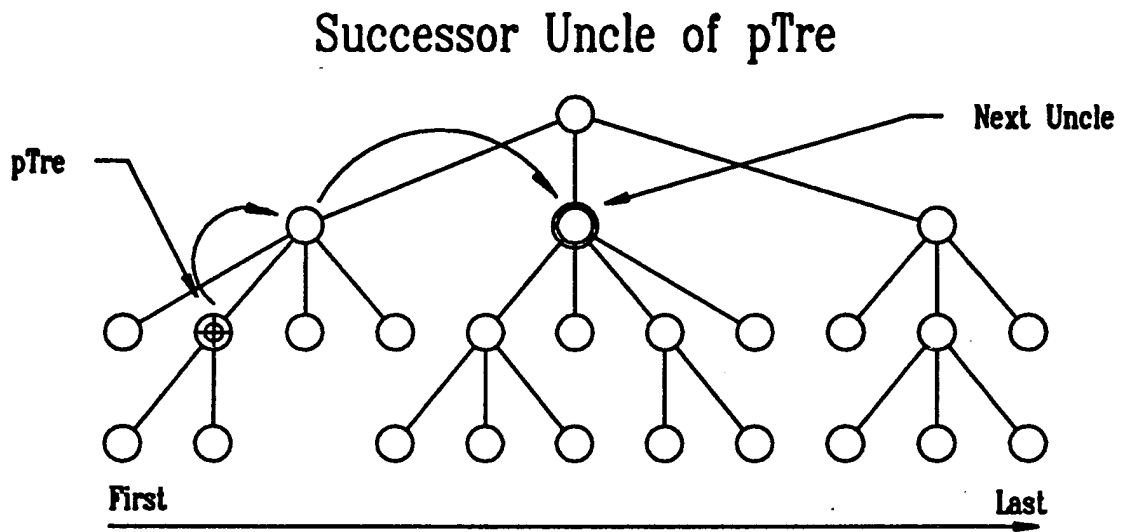
The return value from the TreClientParent function is a pointer to the client of the parent of the *Tree* pTre. NULL is returned if no node exists.

## See Also

TreClient, TreClientFirstChild, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextPreOrder, TreClientNextUncle, TreClientPrev, TreClientPrevPreOrder

# TreClientNextUncle

## Diagram



## TreClientNextUncle

---

### Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientNextUncle( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

### Purpose

The TreClientNextUncle function returns the client of the successor uncle of the *Tree* pTre. The next uncle is found by taking the successor of the parent of pTre. If the parent has no successor the function TreClientNextUncle is called recursively until the root node is reached or an uncle is found.

### Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

### Return Value

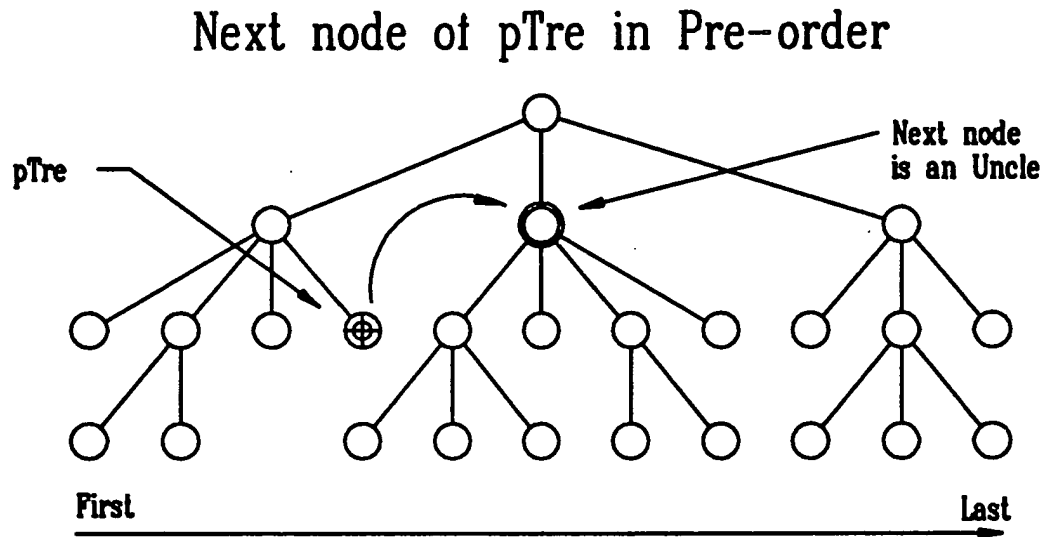
The return value from the TreClientNextUncle function is a pointer to the client of the next uncle of the *Tree* pTre. NULL is returned if there is no uncle node.

### See Also

TreClient, TreClientFirstChild, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextPreOrder, TreClientParent, TreClientPrev, TreClientPrevPreOrder

# TreClientNextPreOrder

## Diagram



# TreClientNextPreOrder

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientNextPreOrder( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientNextPreOrder function returns the client of the pre-order successor to the *Tree* pTre. The pre-order successor is: i) the first child of pTre; ii) the successor sibling of pTre; iii) the uncle of pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

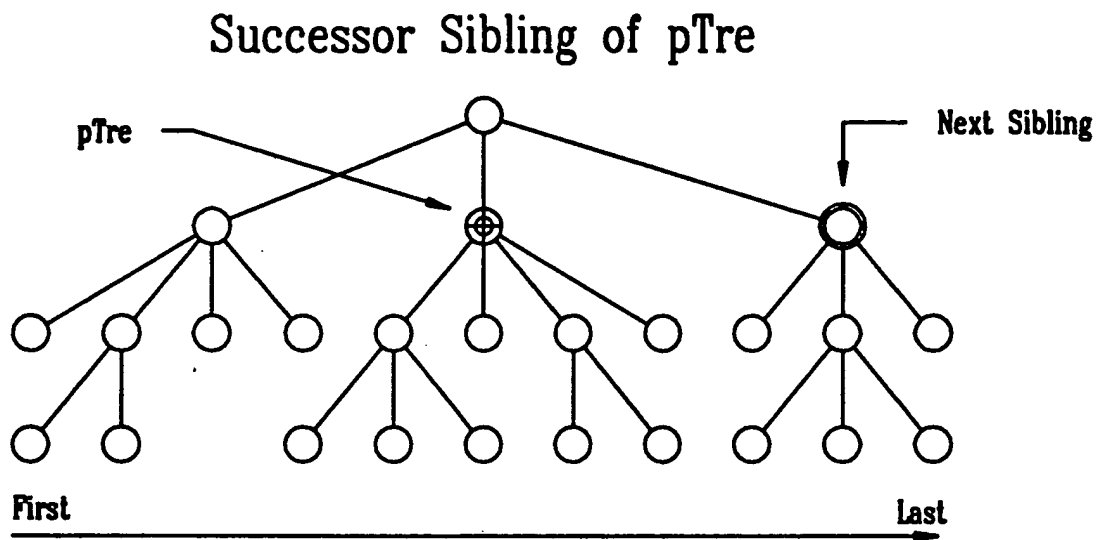
The return value from the TreClientNextPreOrder function is a pointer to the client of the PreOrder successor of the *Tree* pTre. NULL is returned if no PreOrder successor exists.

## See Also

TreClient, TreClientFirstChild, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextUncle, TreClientParent, TreClientPrev, TreClientPrevPreOrder

# TreClientNext

## Diagram



## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientNext( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientNext function returns the client of the successor sibling of the *Tree* pTre.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the TreClientNext function is a pointer to the client of the next sibling of the *Tree* pTre. NULL is returned if there is no sibling.

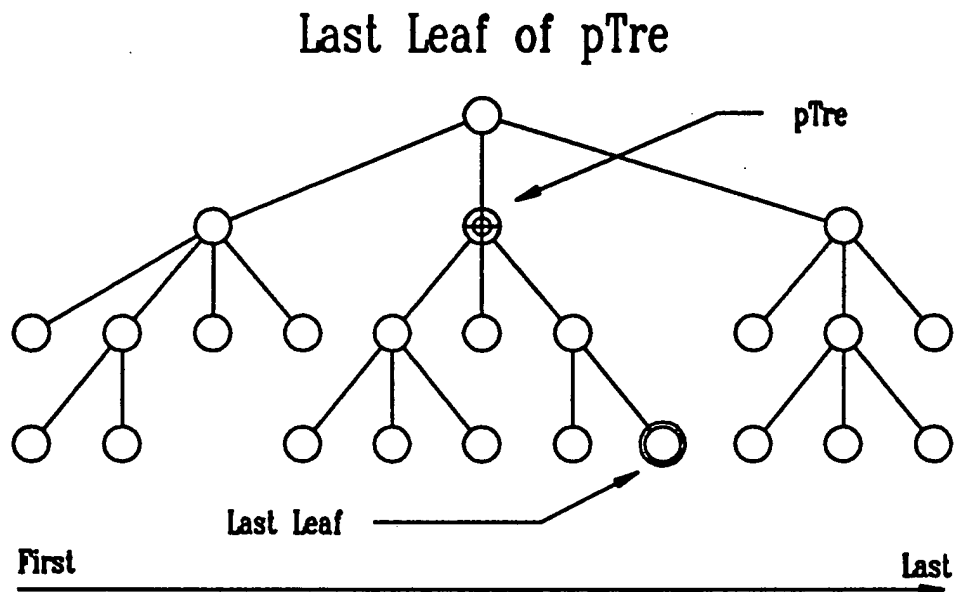
## See Also

TreClient, TreClientFirstChild, TreClientLastChild, TreClientLastLeaf, TreClientNextPreOrder, TreClientNextUncle, TreClientParent, TreClientPrev, TreClientPrevPreOrder



# TreClientLastLeaf

## Diagram



# TreClientLastLeaf

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientLastLeaf( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientLastLeaf function returns the client of the last leaf of the *Tree* pTre. The last leaf is found by calling TreClientLastLeaf recursively for the last child of pTre until a tree node is visited that has no children.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

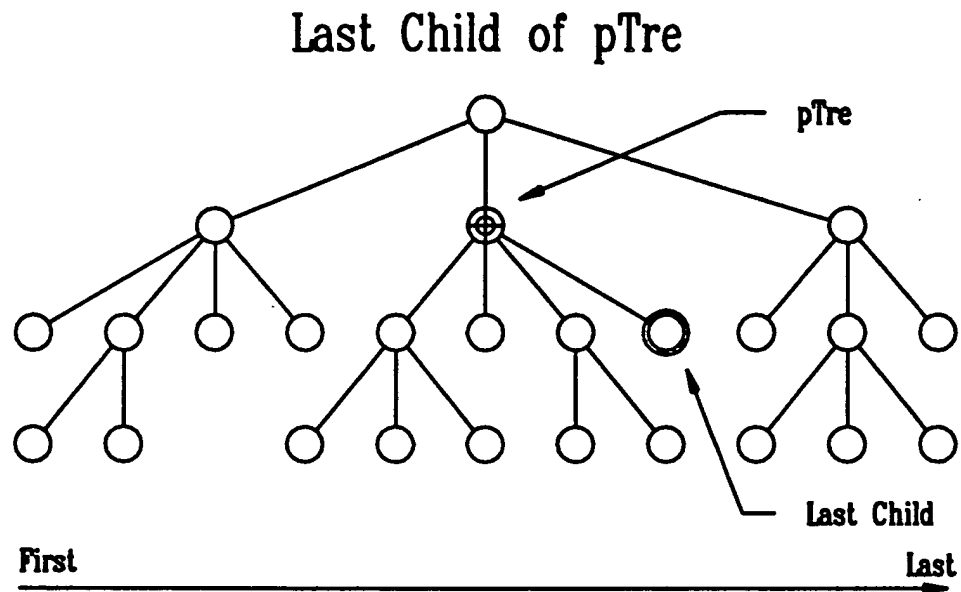
The return value from the TreClientLastLeaf function is a pointer to the client of the last leaf of the *Tree* pTre. NULL is returned if there are no children.

## See Also

TreClient, TreClientFirstChild, TreClientLastChild, TreClientNext, TreClientNextPreOrder, TreClientNextUncle, TreClientParent, TreClientPrev, TreClientPrevPreOrder

# TreClientLastChild

## Diagram



# TreClientLastChild

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientLastChild( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientLastChild function returns the client of the last child of the *Tree* pTre.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

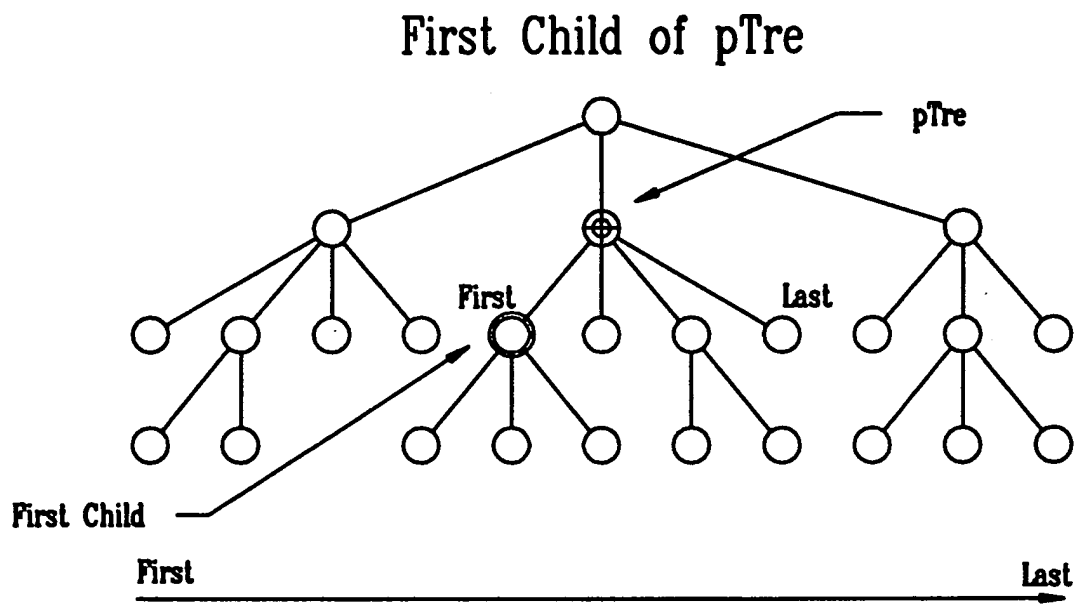
The return value from the TreClientLastChild function is a pointer to the client of the last child of the *Tree* pTre. NULL is returned if there are no children.

## See Also

TreClient, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextPreOrder, TreClientNextUncle, TreClientParent, TreClientPrev, TreClientPrevPreOrder

# TreClientFirstChild

## Diagram



# TreClientFirstChild

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientFirstChild( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClientFirstChild function returns the client of the first child of the *Tree* pTre.

Parameter	-	Description
-----------	---	-------------

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

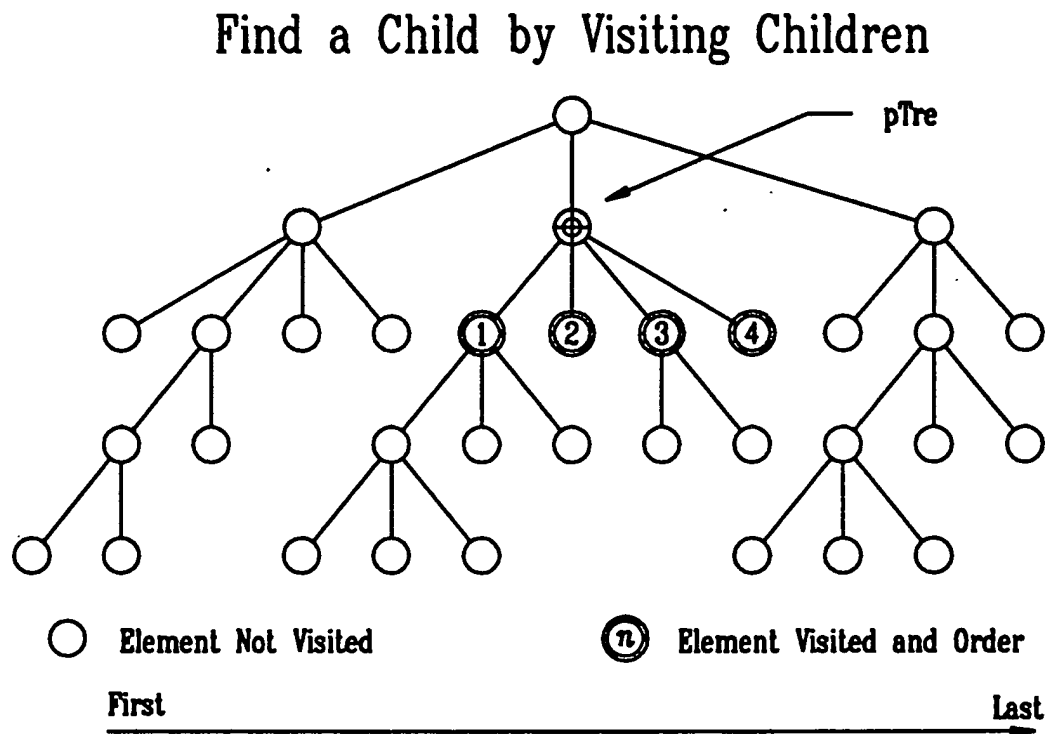
The return value from the TreClientFirstChild function is a pointer to the client of the first child of the *Tree* pTre. NULL is returned if there are no children.

## See Also

TreClient, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextPreOrder, TreClientNextUncle, TreClientParent, TreClientPrev, TreClientPrevPreOrder

# TreClientFindChild

## Diagram



# TreClientFindChild

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClientFindChild( pTre, offset, pBlk )
PTRE      pTre;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The TreClientFindChild function walks a *Tree* and calls the *Tree* client function for each tree node visited. The *Tree* is walked in a forward direction starting with the first child of pTre and ending with the last child of pTre. The function will terminate when the client function returns True or the last child is reached. If the client function returns True a pointer to the client of the tree node is returned otherwise NULL is returned.

The *Block* pBlk contains the client function and an optional list of arguments. The client function must return a boolean (True/False) value.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> . The root of a tree walk.
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the TreClientFindChild function is a pointer to the client of the first child of the *Tree* pTre for which the client function returns True. Otherwise NULL is returned.



# TreClient

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreClient( pTre, offset )
PTRE      pTre;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The TreClient function returns a pointer to the client of the *Tree* pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
offset	-	The distance in bytes between the <i>Tree</i> pTre and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the TreClient function is a pointer to the client of the *Tree* pTre.

## See Also

TreClientFirstChild, TreClientLastChild, TreClientLastLeaf, TreClientNext, TreClientNextPreOrder, TreClientNextUncle, TreClientParent, TreClientPrev, TreClientPrevPreOrder

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreClient function.

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
Void      TreClear( pTre )
PTRE      pTre;
```

Public Function

## Purpose

The TreClear function unlinks and destroys any children of the *Tree* pTre. If pTre is a child node then pTre is cut from the tree.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

No return value

## Notes

[pTre must have no child nodes.]

[pTre must be the root node.]

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreClear function.

# TreAsObj

---

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
POBJ      TreAsObj( pTre )
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The TreAsObj function returns a pointer to the *Object* structure contained by the *Tree* pTre.

Parameter	-	Description
-----------	---	-------------

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TreAsObj function is a pointer to the *Object* structure contained by the *Tree* class.

## Notes

The *Object* pointer can be used to send a message to the client of the *Tree*.

## See Also

TreAsDll; TreAsLel

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreAsObj function.

## Summary

```
#include "cobjects.h"
#include "tremac.h"
```

```
PLEL      TreAsLel( pTre )
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The `TreAsLel` function returns a pointer to the *ListElement* structure contained by the *Tree* `pTre`. The *ListElement* references the successor and predecessor sibling of `pTre`.

## Parameter - Description

---

<code>pTre</code>	-	Pointer to a structure of type <i>Tree</i> .
-------------------	---	--

## Return Value

The return value from the `TreAsLel` function is a pointer to the *ListElement* structure contained by the *Tree* `pTre`. The list element references the successor and predecessor sibling of `pTre`.

## See Also

`TreAsDll`, `TreAsObj`

## Example

Please refer to class test procedure `TSTTRE.C` for an example of the use of the `TreAsLel` function.

# TreAsDll

---

## Summary

```
#include "cobjects.h"  
#include "tremac.h"
```

```
PDLL      TreAsDll( pTre )  
PTRE      pTre;
```

Private Function

A macro is available for this function

## Purpose

The return value from the TreAsDll function is a pointer to the *List* structure contained by the *Tree* pTre. The list contains the children of pTre.

## Parameter - Description

---

pTre	-	Pointer to a structure of type <i>Tree</i> .
------	---	--

## Return Value

The return value from the TreAsDll function is a pointer to the structure of type *List* contained by the *Tree* class. The list contains the child nodes of pTre.

## See Also

TreAsLel, TreAsObj

## Example

Please refer to class test procedure TSTTRE.C for an example of the use of the TreAsDll function.

## Class Reference for *Tree*

<b>Structure Name:</b>	Tree
<b>Abbreviation:</b>	Tre
<b>Class Type:</b>	Inheritable class

# StrToUpper

---

## Example

```
{
char buffer[10];
.
.
/* create a string */
strcpy( buffer, "abcdefghijk" );

/* convert the entire string to upper case */
StrToUpper( buffer );

printf( "buffer = %s\n", buffer );
}
```

The following output will appear to the terminal:  
buffer = ABCDEFGHIJK

# StrToUpper

---

## Summary

```
#include "cobjects.h"  
#include "strmac.h"
```

```
Void          StrToUpper( pStr )  
PSTR          pStr;
```

Public Function

A macro is available for this function

## Purpose

The StrToUpper function changes the case of the *String* pStr to all upper case.

Parameter	-	Description
-----------	---	-------------

---

pStr	-	<i>String</i> to be changed to upper case.
------	---	--

## Return Value

No return value

## Notes

The StrToUpper function uses the C library function toupper().



## StrToMediumInt

---

### Example

```
{
    int i;
    .
    .
    /* create an integer from the string */
    i = StrToMediumInt( "23" );

    printf( "value = %d\n", i );
}
```

The following output will appear to the terminal:  
value = 23

# StrToMediumInt

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
MediumInt    StrToMediumInt( pStr )
PSTR         pStr;
```

Public Function

## Purpose

The StrToMediumInt function returns the numerical representation of the *String* pStr.

Parameter	-	Description
-----------	---	-------------

---

pStr	-	<i>String</i> to be converted to a integer value.
------	---	---

## Return Value

The return value from the StrToMediumInt function is the numerical representation of the *String* pStr.

## Notes

The StrToMediumInt uses the C library function atoi().

## See Also

StrFromMediumInt

# StrToLower

---

## Example

```
{
char buffer[10];

/* create a string */
strcpy( buffer, "ABCDEFGHIJK" );

/* convert the entire string to lower case */
StrToLower( buffer );

printf( "buffer = %s\n", buffer );
}
```

The following output will appear to the terminal:  
buffer = abcdefghijk

# StrToLower

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
Void          StrToLower( pStr )
PSTR          pStr;
```

Public Function

A macro is available for this function

## Purpose

The StrToLower function changes the case of the *String* pStr to all lower case.

## Parameter - Description

---

pStr	-	<i>String</i> to be changed to lower case.
------	---	--

## Return Value

No return value

## Notes

The StrToLower function uses the C library function tolower().

# StrToDate

---

## Notes (cont)

[The length of pStr must be less than 12.]

## See Also

StrFromDate, JulDateStrToJulian, JulValidateDate

## Example

```
{
int  year;
int  month;
int  day;
.
.
/* create year, month, and day from a string */
StrToDate( "12-23-1988", DF_US, &year, &month, &day );

printf( "date %d/%d/%d\n", month, day, year );
}
```

The following output will appear to the terminal:  
date 12/23/1988

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
Void          StrToDate( pStr, format, year, month, day )
PSTR          pStr;
DateFormat    format;
MediumInt     *year;
MediumInt     *month;
MediumInt     *day;
```

Public Function

## Purpose

The StrToDate function converts the *String* pStr, containing a date, into the values of year, month, day. The format of the date string is controlled by passing a date format identifier.

## Parameter - Description

---

pStr	-	Contains the date string of specified format.
format	-	Date string format.
year	-	Returns a 4 digit year. [1583 - 4712]
month	-	Returns a 1-2 digit month. [1 - 12]
day	-	Returns a 1-2 digit day. [1 - 31]

date format:

```
DF_US      > MM-DD-YYYY
DF_EUROPE  > DD-MM-YYYY
DF_MILITARY > YYYY-MM-DD
```

## Return Value

No return value

## Notes

This function is used to parse date strings and is used by the *JulianTime* class. The format can be controlled to any convention by editing the *Class Source* Code file jul.c and examining the sscanf statement within the function.

# StrSqueeze

---

## Example

```
{
char buffer[10];
.
.
/* create a string */
strcpy( buffer, "ABCD12345" );

/* remove a sub-string from a string */
StrSqueeze( buffer, buffer, "123" );

printf( "buffer %s\n", buffer );
}
The following output will appear to the terminal:
buffer ABCD45
```

# StrSqueeze

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
Void          StrSqueeze( pStrD, pStrS, ch )
PSTR          pStrD;
PSTR          pStrS;
Char          ch;
```

Public Function

## Purpose

The StrSqueeze function removes all occurrences of the character *ch* from the *String* *pStr*.

## Parameter - Description

---

pStrD	-	Returns the string with specified character removed.
pStrS	-	<i>String</i> for character removal.
ch	-	Single character to be removed.

## Return Value

No return value



# StrSet

---

## Example

```
{
char str[20];
.
.
/* copy string to another string */
StrSet( str, "ABCD" );

printf( "str = %s\n", str );
}
The following output will appear to the terminal:
str = ABCD
```

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
Void      StrSet( pStrD, pStrS )
PSTR      pStrD;
PSTR      pStrS;
```

Public Function

A macro is available for this function

## Purpose

The StrSet function copies the *String* pStrS to the the *String* pStrD.

Parameter	-	Description
-----------	---	-------------

---

pStrD	-	Returns the string with sub-string replaced.
pStrS	-	Sub-string of original.

## Return Value

No return value

## StrReplaceSubStr

---

### Example

```
{
char buffer[10];
.
.
/* create a string */
strcpy( buffer, "ABCD12345EFGHIJK" );

/* replace a sub-string in a string */
StrReplaceSubStr( buffer, "12345", "XX", 16 );

printf( "buffer %s\n", buffer );
}
The following output will appear to the terminal:
buffer ABCDXXEFGHIJK
```

# StrReplaceSubStr

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
Void          StrReplaceSubStr( pStrOrg, pStrFrom, pStrTo, maxLen )
PSTR          pStrOrg;
PSTR          pStrFrom;
PSTR          pStrTo;
MediumInt     maxLen;
```

Public Function

## Purpose

The StrReplaceSubStr function replaces the sub-string pStrFrom of the *String* pStrOrg with the sub-string pStrTo.

## Parameter - Description

---

pStrOrg	-	Returns the string with sub-string replaced.
pStrFrom	-	Sub-string of original.
pStrTo	-	Sub-string to be used for replacement.
maxLen	-	Maximum length of string.

## Return Value

No return value

## Notes

[maxLen must be in the range [0:128].]

## See Also

StrExtract

# StrInit

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
Void      StrInit( pStr )
PSTR      pStr;
```

Public Function

A macro is available for this function

## Purpose

The StrInit function sets the *String* pStr contents to NULL.

## Parameter - Description

---

pStr	-	A pointer to <i>String</i> .
------	---	------------------------------

## Return Value

No return value

## Example

Please refer to class test procedure TSTSTR.C for an example of the use of the StrInit function.

## StrFromMediumInt

---

### Example

```
{  
char buffer[3];  
.  
.  
/* create a string from an integer */  
StrFromMediumInt( buffer, 2, 23 );  
printf( "buffer %s\n", buffer );  
}
```

The following output will appear to the terminal:  
buffer 23

# StrFromMediumInt

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
MediumInt    StrFromMediumInt( pStr, maxLen, num )
PSTR         pStr;
MediumInt    maxLen;
MediumInt    num;
```

Public Function

## Purpose

The StrFromMediumInt function returns the *String* pStr for the value num.

Parameter	-	Description
-----------	---	-------------

---

pStr	-	Returned string. The string should be able to hold the characters represented by the number plus a NULL terminator.
maxLen	-	Maximum length of string.
num	-	Value to be converted to a string.

## Return Value

The return value from the StrFromMediumInt function is the length of the string returned. The length of pStr is always less than or equal to maxLen.

## See Also

StrToMediumInt

## StrFromDate

---

### See Also

StrToDate, JulJulianToDateStr

### Example

```
{
char dateBuf[11];
.
.
/* create a string from the values of year, month, and day */
StrFromDate( dateBuf, DF_US, 1988, 12, 23 );

printf( "date %s\n", dateBuf );
}
The following output will appear to the terminal:
date 12-23-1988
```



# StrFromDate

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
Void          StrFromDate( pStr, format, year, month, day )
PSTR          pStr;
DateFormat    format;
MediumInt     year;
MediumInt     month;
MediumInt     day;
```

Public Function

## Purpose

The StrFromDate function returns the *String* pStr filled with the date represented by the values year, month, and day. The format of the date string is controlled by passing a date format identifier.

## Parameter - Description

---

pStr	-	Returns the date string. The string should be able to hold 10 characters plus a NULL terminator.
format	-	Date string format.
year	-	A 4 digit year. [1583 - 4712]
month	-	A 1-2 digit month. [1 - 12]
day	-	A 1-2 digit day. [1 - 31]

date format:

```
DF_US      > MM-DD-YYYY
DF_EUROPE  > DD-MM-YYYY
DF_MILITARY > YYYY-MM-DD
```

## Return Value

No return value

## Notes

This function is used to create date strings and is used by the *JulianTime* class. The format can be controlled to any convention by editing the *Class Source* Code file str.c and examining the printf statement within the function.

### Example

```
{  
char buffer[32];  
.  
.  
/* extract a sub-string from a string */  
/* starting at index 4 for 5 characters */  
StrExtract( buffer, "ABCD12345EFGHIJK", 4, 5 );  
  
printf( "buffer = %s\n", buffer );  
}
```

The following output will appear to the terminal:  
buffer = 12345

# StrExtract

---

## Summary

```
#include "cobjects.h"
#include "strmac.h"
```

```
PSTR      StrExtract( pStrD, pStrS, idx, n )
PSTR      pStrD;
PSTR      pStrS;
MediumInt idx;
MediumInt n;
```

Public Function

## Purpose

The StrExtract function extracts the sub-string pStrD from the *String* pStrS starting at an index idx and for a number of characters n.

## Parameter - Description

---

pStrD	-	Returns the extracted sub-string.
pStrS	-	<i>String</i> to be extracted from.
idx	-	Starting index for string extraction. [0 - length]
n	-	Number of characters to be extracted.

## Return Value

The return value from the StrExtract function is a pointer to the string extracted, or NULL.

## See Also

StrReplaceSubStr

## Class Reference for *String*

**Structure Name:** String  
**Abbreviation:** Str  
**Class Type:** Primitive.

## ***Object***

---

This page is intentionally left blank

# ObjSendMessageReturnPtr

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
Void          ObjSendMessageReturnPtr( pObj, m, pBlk )
POBJ          pObj;
MediumInt     m;
PBLK          pBlk;
```

Public Function

## Purpose

The ObjSendMessageReturnPtr function sends the message identified by its selector index *m* to the *Object* pObj. The *Block* pBlk can be used to pass parameters to the method implementing the message. The return value is the value returned by the method.

## Parameter - Description

---

pObj	-	Pointer to a structure of type <i>Object</i> .
m	-	The message selector index.
pBlk	-	Pointer to a structure of type <i>Block</i> .

## Return Value

No return value

## Notes

[The message number must be greater than or equal to zero, and less than the number of messages.]

## See Also

ObjSendMessage, ObjSendMessageReturnInt

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjSendMessageReturnPtr function.

# ObjSendMessageReturnInt

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
MediumInt    ObjSendMessageReturnInt( pObj, m, pBlk )
POBJ         pObj;
MediumInt    m;
PBLK         pBlk;
```

Public Function

## Purpose

The ObjSendMessageReturnInt function sends the message identified by its selector index *m* to the *Object* *pObj*. The *Block* *pBlk* can be used to pass parameters to the method implementing the message. The return value is the value returned by the method.

## Parameter - Description

---

pObj	-	Pointer to a structure of type <i>Object</i> .
m	-	The message selector index.
pBlk	-	Pointer to a structure of type <i>Block</i> .

## Return Value

The return value from the ObjSendMessageReturnInt function is the value returned by the method which implements the message.

## Notes

[The message number must be greater than or equal to zero, and less than the number of messages.]

## See Also

ObjSendMessage, ObjSendMessageReturnPtr

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjSendMessageReturnInt function.

# ObjSendMessage

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
Void          ObjSendMessage( pObj, m, pBlk )
POBJ          pObj;
MediumInt     m;
PBLK          pBlk;
```

Public Function

## Purpose

The ObjSendMessage function sends the message identified by its selector index *m* to the *Object* *pObj*. The *Block* *pBlk* can be used to pass parameters to the method implementing the message.

## Parameter - Description

---

pObj	-	Pointer to a structure of type <i>Object</i> .
m	-	The message selector index.
pBlk	-	Pointer to a structure of type <i>Block</i> .

## Return Value

No return value

## Notes

[The message number must be greater than or equal to zero, and less than the number of messages.]

## See Also

ObjSendMessageReturnInt, ObjSendMessageReturnPtr

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjSendMessage function.



# ObjRespondsToSelector

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
MediumInt    ObjRespondsToSelector( pObj, pStr )
POBJ          pObj;
PSTR          pStr;
```

Public Function

## Purpose

The ObjRespondsToSelector function returns the index of the message selector if the *Object* pObj can respond to the selector *String* pointer pStr. If the *Object* pObj does not respond to the message, -1 is returned.

## Parameter - Description

---

pObj	-	Pointer to a structure of type <i>Object</i> .
pStr	-	Pointer to <i>String</i> which contains the selector name

## Return Value

The return value from the ObjRespondsToSelector function is True if pObj responds to the selector and False otherwise.

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjRespondsToSelector function.

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
Bool      ObjIsRoot( pObj )
POBJ      pObj;
```

Public Function

## Purpose

The ObjIsRoot function returns True if the *Object* pObj is a root *Object*, that is, it does not have a subobject.

## Parameter - Description

---

pObj - Pointer to a structure of type *Object*.

## Return Value

The return value from the ObjIsRoot function True if pObj is a root object or False otherwise.

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjIsRoot function.

# ObjInit

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
Void      ObjInit( pObj, pCls )
POBJ      pObj;
PCLS      pCls;
```

Public Function

## Purpose

The ObjInit function initializes the *Object* pObj to be an object of *Class* pCls.

## Parameter - Description

---

pObj	-	Pointer to a structure of type <i>Object</i> .
pCls	-	Pointer to a structure of type <i>Class</i> .

## Return Value

No return value

## Notes

If pObj was created with ClsCreateObject it does not need to be initialized since ObjInit is called automatically.

If pObj was statically allocated, ObjInit should be the first function called.

## See Also

ClsCreateObject, ObjDeInit, ObjDestroy

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjInit function.

# ObjGetSubObjectOffset

---

## Summary

```
#include "cobjects.h"  
#include "objmac.h"
```

```
MediumInt    ObjGetSubObjectOffset( pObj )  
POBJ         pObj;
```

Public Function

## Purpose

The ObjGetImmediateClientOffset function returns the offset of the immediate client.

Parameter	-	Description
-----------	---	-------------

pObj	-	Pointer to a structure of type <i>Object</i> .
------	---	--

## Return Value

The return value from the ObjGetImmediateClientOffset function is the offset of the immediate client object.

## See Also

ObjGetImmediateClient, ObjGetClient, ObjGetRootClient

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetSubObjectOffset function.

# ObjGetSize

---

## Summary

```
#include "cobjects.h"  
#include "objmac.h"
```

```
MediumInt    ObjGetSize( pObj )  
POBJ         pObj;
```

Public Function

## Purpose

The ObjGetSize function returns the size of the *Object* pObj.

## Parameter - Description

---

pObj - Pointer to a structure of type *Object*.

## Return Value

The return value from the ObjGetSize function is the size of pObj.

## See Also

ObjGetRootObjectSize

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetSize function.

# ObjGetRootClientSize

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
MediumInt    ObjGetRootClientSize( pObj )
POBJ         pObj;
```

Public Function

## Purpose

The ObjGetRootClientSize function returns the size of the root *Object* of the *Object* pObj.

Parameter	-	Description
-----------	---	-------------

pObj	-	Pointer to a structure of type <i>Object</i> .
------	---	--

## Return Value

The return value from the ObjGetRootClientSize function is the size of the root *Object* of pObj.

## See Also

ObjGetSize

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetRootClientSize function.

# ObjGetRootClient

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
POBJ          ObjGetRootClient( pObj )
POBJ          pObj;
```

Public Function

## Purpose

The ObjGetRootClient function returns the Root client object of the *Object* pObj.

Parameter	-	Description
-----------	---	-------------

---

pObj	-	Pointer to a structure of type <i>Object</i> .
------	---	--

## Return Value

The return value from the ObjGetRootClient function is a pointer to an *Object*.

## Notes

The root client is the client *Object* which has no subobjects.

## See Also

ObjGetImmediateClient, ObjGetClient

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetRootClient function.

# ObjGetNthSuperObject

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
POBJ      ObjGetNthSuperObject( pObj, n )
POBJ      pObj;
MediumInt n;
```

Public Function

## Purpose

The ObjGetNthSuperObject function returns the Nth superobject of the *Object* pObj.

Parameter	-	Description
-----------	---	-------------

---

pObj	-	Pointer to a structure of type <i>Object</i> .
n	-	The index of the superobject.

## Return Value

The return value from the ObjGetNthSuperClient function is a pointer to an *Object*.

## Notes

[The super class index must be greater than or equal to zero, and less than the number of superclasses.]

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetNthSuperObject function.



# ObjGetMethodAndOffset

---

## Example

Please refer to class test procedure TSTOBJ:C for an example of the use of the ObjGetMethodAndOffset function.

# ObjGetMethodAndOffset

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
Void      ObjGetMethodAndOffset( pObj, m, ppMth, pOffset )
POBJ      pObj;
MediumInt m;
PMTH      *ppMth;
MediumInt *pOffset;
```

Public Function

## Purpose

The ObjGetMethodAndOffset function returns a pointer to the Method which implements the message identified by the message selector index *m* for the *Object* *pObj*. It also returns the offset which will determine the object to be passed to the method. The Method pointer is returned in *ppMth* and the offset in *pOffset*.

## Parameter - Description

---

<i>pObj</i>	-	Pointer to a structure of type <i>Object</i> .
<i>m</i>	-	The message selector index.
<i>ppMth</i>	-	Pointer to a pointer to Method.
<i>pOffset</i>	-	Pointer to a MediumInt.

range: \**pOffset* will be in the range [-2000:2000]. A negative value means a client of *pObj* is overriding the message. A positive value means *pObj* is inheriting the method from a superobject. A zero value is ambiguous and could be a client overriding the method or the method may be implemented by *pObj*.

## Return Value

No return value

## Notes

*m* must be in the range [0:N-1] where *N* is the number of objects *pObj* can respond to.

[The message number must be greater than or equal to zero, and less than the number of messages.]

# ObjGetImmediateClient

---

## Summary

```
#include "cobjects.h"  
#include "objmac.h"
```

```
POBJ      ObjGetImmediateClient( pObj )  
POBJ      pObj;
```

Public Function

## Purpose

The ObjGetImmediateClient function returns the immediate client *Object* of the *Object* pObj. The immediate client is the object which directly subclasses pObj.

## Parameter - Description

---

pObj - Pointer to a structure of type *Object*.

## Return Value

The return value from the ObjGetImmediateClient function is a pointer to an *Object*.

## See Also

ObjGetClient

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetImmediateClient function.

# ObjGetClsName

---

## Summary

```
#include "cobjects.h"  
#include "objmac.h"
```

```
PSTR      ObjGetClsName( pObj )  
POBJ      pObj;
```

Public Function

## Purpose

The ObjGetClassName function returns a pointer to a *String* which contains the name of the *Class* which defines the *Object* pObj.

## Parameter - Description

---

pObj - Pointer to a structure of type *Object*.

## Return Value

The return value from the ObjGetClassName function is a pointer to a *String* containing the class name

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetClsName function.

# ObjGetCls

---

## Summary

```
#include "cobjects.h"  
#include "objmac.h"
```

```
PCLS      ObjGetCls( pObj )  
POBJ      pObj;
```

Public Function

## Purpose

The ObjGetCls function returns a pointer to a structure of type *Class* which is the *Class* of the *Object* pObj.

## Parameter - Description

---

pObj	-	Pointer to a structure of type <i>Object</i> .
------	---	--

## Return Value

The return value from the ObjGetCls function is a pointer to a structure of type *Class* which is the *Class* of pObj.

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetCls function.

## ObjGetClientOrNull

---

### Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetClientOrNull function.

# ObjGetClientOrNull

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
POBJ      ObjGetClientOrNull( pObj, offset )
POBJ      pObj;
MediumInt offset;
```

Public Function

## Purpose

The ObjGetClientOrNull function returns the client *Object* of the *Object* pObj located at offset. If pObj is NULL, NULL is returned.

## Parameter - Description

---

pObj	-	Pointer to a structure of type <i>Object</i> .
offset	-	The distance in bytes between the <i>Object</i> pObj and it's client pointer.

## Return Value

The return value from the ObjGetClientOrNull function is a pointer to an *Object*.

## Notes

The offset is calculated when the *Class* which defines pObj was created. See the section on *Classes* and *MetaClasses* for more details.

The range for offset is [-2000:2000]. If the offset is zero or negative it means the object being retrieved is a subobject (or client) of pObj. If the offset is positive, the object being retrieved is a superobject of pObj.

## See Also

ObjGetClient

## ObjGetClient

---

### Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjGetClient function.



# ObjGetClient

---

## Summary

```
#include "cobjects.h"
#include "objmac.h"
```

```
POBJ      ObjGetClient( pObj, offset )
POBJ      pObj;
MediumInt offset;
```

Public Function

## Purpose

The ObjGetClient function returns the client *Object* of the *Object* pObj located at offset.

Parameter	-	Description
-----------	---	-------------

pObj	-	Pointer to a structure of type <i>Object</i> .
offset	-	The distance in bytes between the <i>Object</i> pObj and it's client pointer.

## Return Value

The return value from the ObjGetClient function is a pointer to an *Object*.

## Notes

The offset is calculated when the *Class* which defines pObj was created. See the section on Classes and MetaClasses for more details.

The range for offset is [-2000:2000]. If the offset is zero or negative it means the object being retrieved is a subobject (or client) of pObj. If the offset is positive, the object being retrieved is a superobject of pObj.

## See Also

ObjGetClientOrNull

# ObjDestroy

---

## Summary

```
#include "cobjects.h"  
#include "objmac.h"
```

```
Void          ObjDestroy( pObj )  
POBJ          pObj;
```

Public Function

## Purpose

The ObjDestroy function deinitializes and deallocates the *Object* pObj. The *Object* pObj should not be referenced after this call since the memory will have been freed.

## Parameter - Description

---

pObj - Pointer to a structure of type *Object*.

## Return Value

No return value

## Notes

If the object was created from dynamic memory via ClsCreateObject, then the function ObjDestroy should be called instead of ObjDeInit.

If the object was created from statically allocated memory, it should have been initialized via ObjInit in which case ObjDeInit should be used to deinitialize it.

## See Also

ClsCreateObject, ObjDestroy, ObjInit

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjDestroy function.

# ObjDeInit

---

## Summary

```
#include "cobjects.h"  
#include "objmac.h"
```

```
Void      ObjDeInit( pObj )  
POBJ      pObj;
```

Public Function

## Purpose

The ObjDeInit function deinitializes the *Object* pObj.

## Parameter - Description

---

pObj - Pointer to a structure of type *Object*.

## Return Value

No return value

## Notes

If the object was created from dynamic memory via ClsCreateObject, then the function ObjDestroy should be called instead of ObjDeInit.

If the object was created from statically allocated memory, it should have been initialized via ObjInit in which case ObjDeInit should be used to deinitialize it.

The ObjDeInit function should be the last reference to pObj.

## See Also

ClsCreateObject, ObjDestroy, ObjInit

## Example

Please refer to class test procedure TSTOBJ.C for an example of the use of the ObjDeInit function.

## Class Reference for *Object*

**Structure Name:** Object  
**Abbreviation:** Obj  
**Class Type:** Primitive Class.

# MsgSendReturnPtr

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
Void      MsgSendReturnPtr( pMsg, pObj, pBlk )
PMSG      pMsg;
POBJ      pObj;
PBLK      pBlk;
```

Public Function

## Purpose

The MsgSendReturnPtr function sends the *Message* pMsg to the *Object* pObj and passes the *Block* pBlk parameters to the function which implements the message. The return value is the value returned by the function implementing the message.

## Parameter - Description

---

pMsg	-	Pointer to a structure of type <i>Message</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .
pBlk	-	Pointer to a structure of type <i>Block</i> .

## Return Value

No return value

# MsgSendReturnInt

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
MediumInt    MsgSendReturnInt( pMsg, pObj, pBlk )
PMSG         pMsg;
POBJ         pObj;
PBLK         pBlk;
```

Public Function

## Purpose

The MsgSendReturnInt function sends the *Message* pMsg to the *Object* pObj and passes the *Block* pBlk parameters to the function which implements the message. The return value is the value returned by the function implementing the message.

Parameter	-	Description
-----------	---	-------------

---

pMsg	-	Pointer to a structure of type <i>Message</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .
pBlk	-	Pointer to a structure of type <i>Block</i> .

## Return Value

The return value from the MsgSendReturnInt function is a the value returned by the method.

# MsgSend

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
Void      MsgSend( pMsg, pObj, pBlk )
PMSG      pMsg;
POBJ      pObj;
PBLK      pBlk;
```

Public Function

## Purpose

The MsgSend function sends the *Message* pMsg to the *Object* pObj and passes the *Block* pBlk parameters to the function which implements the message.

## Parameter - Description

---

pMsg	-	Pointer to a structure of type <i>Message</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .
pBlk	-	Pointer to a structure of type <i>Block</i> .

## Return Value

No return value

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
Void          MsgPrint( pMsg, pCio, item, level, name )
PMSG          pMsg;
PCIO          pCio;
MediumInt     item;
MediumInt     level;
PSTR          name;
```

Public Function

## Purpose

The MsgPrint function prints the contents of the *Message* pMsg on the ConsoleInputOutput device pCio. The item parameter is the array index of this instance or -1 if it is not an array element, level is a number indicating the level of indentation, and name is a *String* pointer which is the name of this instance.

## Parameter - Description

---

pMsg	-	Pointer to a structure of type <i>Message</i> .
pCio	-	Pointer to a structure of type ConsoleInputOutput.
item	-	The array element number of this instance (or -1).
level	-	the level of indentation to print this object with.
name	-	the name of this instance.

## Return Value

No return value



# MsgInit

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
Void      MsgInit( pMsg, pCls, pMms )
PMSG      pMsg;
PCLS      pCls;
PMMS      pMms;
```

Public Function

## Purpose

The MsgInit function initializes the *Message* pMsg of the *Class* pCls with the *MetaMessage* pMms.

## Parameter - Description

---

pMsg	-	Pointer to a structure of type <i>Message</i> .
pCls	-	Pointer to a structure of type <i>Class</i> .
pMms	-	Pointer to a structure of type <i>MetaMessage</i> .

## Return Value

No return value

# MsgGetSelector

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
PSTR      MsgGetSelector( pMsg )
PMSG      pMsg;
```

Public Function

## Purpose

The MsgGetSelector function returns a pointer to a *String* containing the selector name.

## Parameter - Description

---

pMsg - Pointer to a structure of type *Message*.

## Return Value

The return value from the MsgGetSelector function is a pointer to a *String* containing the selector name.

# MsgGetMethod

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
PMTH      MsgGetMethod( pMsg )
PMSG      pMsg;
```

Public Function

## Purpose

The MsgGetMethod function returns a pointer to a Method which implements the *Message* pMsg.

## Parameter - Description

---

pMsg	-	Pointer to a structure of type <i>Message</i> .
------	---	---

## Return Value

The return value from the MsgGetMethod function is a pointer to a Method which implements pMsg.

# MsgGetOffset

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
MediumInt    MsgGetOffset( pMsg )
PMSG         pMsg;
```

Public Function

## Purpose

The MsgGetOffset function returns the *Object* offset for the *Message* pMsg.

## Parameter - Description

---

pMsg	-	Pointer to a structure of type <i>Message</i> .
------	---	---

## Return Value

The return value from the MsgGetOffset function is the *Object* offset for this message.

# MsgDeInit

---

## Summary

```
#include "cobjects.h"
#include "msgmac.h"
```

```
Void      MsgDeInit( pMsg )
PMSG      pMsg;
```

Public Function

## Purpose

The MsgDeInit function deinitializes the *Message* pMsg. The MsgDeInit function should be the last function called when using the *Message* pMsg.

## Parameter - Description

---

pMsg - Pointer to a structure of type *Message*.

## Return Value

No return value

## See Also

MsgDestroy, MsgInit

## Class Reference for *Message*

<b>Structure Name:</b>	Message
<b>Abbreviation:</b>	Msg
<b>Class Type:</b>	Primitive Class

# MscValidate

---

## Summary

```
#include "cobjects.h"  
#include "mscmac.h"
```

```
Void      MscValidate( pMsc )  
PMSC      pMsc;
```

Public Function

## Purpose

The MscValidate function verifies that the *MetaSuperClass* pMsc is a valid instance.

Parameter	-	Description
-----------	---	-------------

---

pMsc	-	Pointer to a structure of type <i>MetaSuperClass</i> .
------	---	--

## Return Value

No return value

## Notes

Because instances of type *MetaSuperClass* are typically statically defined and initialized at compile time, this function provides a way of verifying that an instance was initialized properly.

This function is called by McIValidate.

An invalid instance will generate an exception.

## See Also

McIValidate, MmsValidate

## Summary

```
#include "cobjects.h"
#include "mscmac.h"
```

```
Void      MscPrint( pMsc, pCio, item, level, name )
PMSC      pMsc;
PCIO      pCio;
MediumInt item;
MediumInt level;
PSTR      name;
```

Public Function

## Purpose

The MscPrint function displays the contents of the *MetaSuperClass* pMsc on the ConsoleInputOutput device pCio. Item identifies if pMsc is part of an array, level indicates the level of indenting to use when printing the contents, and name indicates the name of this instance.

## Parameter - Description

---

pMsc	-	Pointer to a structure of type <i>MetaSuperClass</i> .
pCio	-	Pointer to a structure of type ConsoleInputOutput.
item	-	The array index of this instance or -1.
level	-	The nesting level of this instance.
name	-	Pointer to a <i>String</i> which names this instance.

## Return Value

No return value



# MscGetOffset

---

## Summary

```
#include "cobjects.h"  
#include "mscmac.h"
```

```
MediumInt    MscGetOffset( pMsc )  
PMSC         pMsc;
```

Public Function

## Purpose

The MscGetOffset function returns the offset of a superobject from an object for the MetaSuperClas pMsc.

## Parameter - Description

---

pMsc - Pointer to a structure of type *MetaSuperClass*.

## Return Value

The return value from the MscGetOffset function is the offset of the superobject from the object.

## Notes

The offset is used by an object to retrieve its client(s), also called subobjects.

A side effect of this function is to calculate the offset and store it in the offset pointed to by this instance.

[The superobject offset must be in the range [0:2000].]

# MscGetMetaClass

---

## Summary

```
#include "cobjects.h"
#include "mscmac.h"
```

```
PMCL      MscGetMetaClass( pMsc )
PMSC      pMsc;
```

Public Function

## Purpose

The MscGetMetaClass function returns the pointer to a structure of type *MetaClass* for the *MetaSuperClass* pMsc.

## Parameter - Description

---

pMsc - Pointer to a structure of type *MetaSuperClass*.

## Return Value

The return value from the MscGetMetaClass function is a pointer to a structure of type *MetaClass* which is the *MetaClass* being inherited.

## See Also

MscGetName

# MscGetName

---

## Summary

```
#include "cobjects.h"  
#include "mscmac.h"
```

```
PSTR      MscGetName( pMsc )  
PMSC      pMsc;
```

Public Function

## Purpose

The MscGetName function returns a *String* pointer which uniquely identifies the *MetaSuperClass* pMsc.

## Parameter - Description

---

pMsc	-	Pointer to a structure of type <i>MetaSuperClass</i> .
------	---	--

## Return Value

The return value from the MscGetName function is a *String* pointer to the name of the superclass

## Notes

There must always be a superclass name.

Superclasses are typically named the same as the structure they refer to. When a class inherits from two or more superclasses of the same type, the superclass name is prefixed with a name describing its use.

## See Also

MscGetMetaClass

## Class Reference for *MetaSuperClass*

**Structure Name:** MetaSuperClass

**Abbreviation:** Msc

**Class Type:** Primitive Class

# MmsValidate

---

## Summary

```
#include "cobjects.h"  
#include "mmsmac.h"
```

```
Void      MmsValidate( pMms )  
PMMS      pMms;
```

Public Function

## Purpose

The MmsValidate function determines whether or not the instance pMms is a valid, properly initialized *MetaMessage*.

## Parameter - Description

---

pMms - Pointer to a structure of type *MetaMessage*.

## Return Value

No return value

## Notes

Because instances of *MetaMessage* are typically initialized at compile time, this function can be used to ensure that it was initialized properly.

An invalid pMms will generate an exception.

## Summary

```
#include "cobjects.h"
#include "mmsmac.h"
```

```
Void      MmsPrint( pMms, pCio, item, level, name )
PMMS      pMms;
PCIO      pCio;
MediumInt item;
MediumInt level;
PSTR      name;
```

Public Function

## Purpose

The MmsPrint function displays the contents of the *MetaMessage* pMms on the ConsoleInputOutput device pCio. Item identifies if pMms is part of an array, level indicates the level of indenting to use when printing the contents, and name indicates the name of this instance.

## Parameter - Description

---

pMms	-	Pointer to a structure of type <i>MetaMessage</i> .
pCio	-	Pointer to a structure of type ConsoleInputOutput.
item	-	The array index of this instance or -1.
level	-	The nesting level of this instance.
name	-	Pointer to a <i>String</i> which names this instance.

## Return Value

No return value

# MmsGetSelector

---

## Summary

```
#include "cobjects.h"  
#include "mmsmac.h"
```

```
PSTR      MmsGetSelector( pMms )  
PMMS      pMms;
```

Public Function

## Purpose

The MmsGetSelector function returns a *String* pointer which identifies the *MetaMessage* pMms.

## Parameter - Description

---

pMms - Pointer to a structure of type *MetaMessage*.

## Return Value

The return value from the MmsGetSelector function is the name of the selector for this message

## Notes

It is up to the user to make sure the selector name is initialized properly.

There must always be a selector given.

Selectors are typically named similarly to functions, but without the structure name prefix, and with the first letter lowercased.

## See Also

MmsGetSuper, MmsGetSelector

### Notes (cont)

If the superclass is non-NULL and the method is non-NULL, this indicates the message is being overridden. It is verified during *Class* creation (MclCreateClass) that the superclass name is valid and that the named superclass does in fact respond to the selector in this message.

Typically, the name of the superclass is specified as the full structure name to which it refers.

### See Also

MmsGetSuper, MmsGetSelector



# MmsGetSuper

---

## Summary

```
#include "cobjects.h"
#include "mmsmac.h"
```

```
PSTR      MmsGetSuper( pMms )
PMMS      pMms;
```

Public Function

## Purpose

The MmsGetSuper function returns the *String* pointer which names the superclass the *MetaMessage* pMms inherits from. If this is a new message not inherited from any subclasses, the function returns NULL.

## Parameter - Description

---

pMms - Pointer to a structure of type *MetaMessage*.

## Return Value

The return value from the MmsGetSuper function is the name of the superclass this message is inherited from or NULL if it is not inherited.

## Notes

It is up to the user to initialize this value properly. Instances of this class are always associated with a particular *MetaClass* instance. The name of the superclass must match precisely the name of one of the superclasses in the array of *MetaSuperClasses* contained in the *MetaClass* instance.

If the method is NULL and so is the superclass name, this indicates that it is the responsibility of a subclass to implement this function. Failure to do so is detected at *Class* creation time (*MclCreateClass*).

If the method is NULL, then this indicates the method is to be inherited from a superclass of the *MetaClass* which contains this *MetaMessage*.

If the superclass name is NULL but the method is non-NULL, this indicates the message is unique to this class and not inherited from a superclass. It is up to the user to ensure this is true. Forthcoming versions of this code will verify this.

## MmsGetMethod

---

### See Also

MmsGetSuper, MmsGetSelector

# MmsGetMethod

---

## Summary

```
#include "cobjects.h"  
#include "mmsmac.h"
```

```
PMTH      MmsGetMethod( pMms )  
PMMS      pMms;
```

Public Function

## Purpose

The MmsGetMethod function returns the Method (function) pointer for the *MetaMessage* pMms or NULL if the method is to be inherited from a superclass.

## Parameter - Description

---

pMms - Pointer to a structure of type *MetaMessage*.

## Return Value

The return value from the MmsGetMethod function is a (function) pointer to the Method associated with the *MetaMessage* pMms or NULL.

## Notes

If the method returned is NULL, then this indicates the method is to be inherited from a superclass of the *MetaClass* which contains this *MetaMessage*.

If the method is NULL and so is the superclass name, this indicates that it is the responsibility of a subclass to implement this function. Failure to do so is detected at *Class* creation time (MclCreateClass).

If the method is non-NULL but the superclass is NULL, this indicates the message is unique to this class and not inherited from a superclass. It is up to the user to ensure this is true. Forthcoming versions of this code will verify this.

If the method is non-NULL and the superclass is non-NULL, this indicates the message is being overridden. It is verified during *Class* creation (MclCreateClass) that the superclass name is valid and that the named superclass does in fact respond to the selector in this message.

## Class Reference for *MetaMessage*

<b>Structure Name:</b>	MetaMessage
<b>Abbreviation:</b>	Mms
<b>Class Type:</b>	Primitive Class

# MemSetChr

---

## Summary

```
#include "cobjects.h"
#include "memmac.h"
```

```
Void      MemSetChr( pMem, s, i, n, os, c )
Mem      *pMem;
UMediumInt s;
UMediumInt i;
UMediumInt n;
UMediumInt os;
Chr      c;
```

Public Function

## Purpose

The MemSetChr function sets a region of the array pMem to the Char c. The size of the array in elements is s, the beginning index is i, the number of elements to clear is n, and the size of each element is os.

## Parameter - Description

---

pMem	-	Pointer to <i>Memory</i> .
s	-	Size of pMem in elements.
i	-	First array index to start with.
n	-	Number of elements to clear.
os	-	Size of each element.
c	-	The character to set the region to

## Return Value

No return value

## Notes

[i must be less s.]

[(i + n) must be less or equal to s.]

[os must be greater than 0.]

## Summary

```
#include "cobjects.h"
#include "memmac.h"
```

```
Void      MemPasteNSet( pMem, s, i, n, os, c )
Mem      *pMem;
UMediumInt s;
UMediumInt i;
UMediumInt n;
UMediumInt os;
Chr      c;
```

Public Function

## Purpose

The MemPasteNSet function clears a region of the array pMem to zeros. The range of elements at from i through s are copied past the last element of the region being cleared. This results in the last n elements of the *Memory* pMem being copied over. The size of the array in elements is s, the beginning index is i, the number of elements to clear is n, and the size of each element is os.

Parameter	-	Description
-----------	---	-------------

---

pMem	-	Pointer to <i>Memory</i> .
s	-	Size of pMem in elements.
i	-	First array index to start with.
n	-	Number of elements to clear.
os	-	Size of each element.

## Return Value

No return value

## Notes

[i must be less s.]

[(i + n) must be less or equal to s.]

[os must be greater than 0.]

# MemNew

---

## Summary

```
#include "cobjects.h"  
#include "memmac.h"
```

```
PMEM      MemNew( amount )  
UMediumInt amount;
```

Public Function

## Purpose

The MemNew function returns newly allocated *Memory* of amount bytes. If the memory cannot be allocated, NULL is returned.

## Parameter - Description

---

amount	-	The amount to allocate
--------	---	------------------------

## Return Value

The return value from the function MemNew is a pointer to *Memory* which was allocated or NULL if it could not be allocated.

# MemDuplicate

---

## Summary

```
#include "cobjects.h"  
#include "memmac.h"
```

```
Void      MemDuplicate( pMem, s, pMemS, os )  
Mem      *pMem;  
UMediumInt s;  
Mem      *pMemS;  
UMediumInt os;
```

Public Function

## Purpose

The MemDuplicate function copies the *Memory* pMemS to the *Memory* pMem. The size of the *Memory* to copy in elements is s and the size of each element is os.

## Parameter - Description

---

pMem	-	Pointer to <i>Memory</i> .
s	-	Size of pMem in elements.
pMemS	-	Pointer to <i>Memory</i> .
os	-	Size of each element.

## Return Value

No return value

## Notes

[os must be greater than 0.]



# MemDestroy

---

## Summary

```
#include "cobjects.h"
#include "memmac.h"
```

```
Void      MemDestroy( pMem )
PMEM      pMem;
```

Public Function

## Purpose

The MemDestroy function deallocates the *Memory* pMem previously allocated by MemNew.

Parameter	-	Description
-----------	---	-------------

---

pMem	-	Pointer to <i>Memory</i> .
------	---	----------------------------

## Return Value

No return value

## Summary

```
#include "cobjects.h"
#include "memmac.h"
```

```
Void      MemCutNSet( pMem, s, i, n, os, c )
Mem      *pMem;
UMediumInt s;
UMediumInt i;
UMediumInt n;
UMediumInt os;
Chr      c;
```

Public Function

## Purpose

The MemCutNSet function deletes a region of the array pMem and copies elements following the region on top of the region being cut. The size of the array in elements is s, the beginning of the region to cut is i, the number of elements to cut is n, the size of each element is os, and c is the value to copy to the open positions at the end of the array.

Parameter	-	Description
-----------	---	-------------

---

pMem	-	Pointer to <i>Memory</i> .
s	-	Size of pMem in elements.
i	-	First array index to start with.
n	-	Number of elements to cut.
os	-	Size of each element.

## Return Value

No return value

## Notes

[i must be less s.]

[(i + n) must be less or equal to s.]

[os must be greater than 0.]

# MemCopy

---

## Notes (cont)

[(di + n) must be less or equal to s.]

## Summary

```
#include "cobjects.h"
#include "memmac.h"
```

```
Void          MemCopy( pMem, s, di, n, i, os )
Mem          *pMem;
UMediumInt   s;
UMediumInt   di;
UMediumInt   n;
UMediumInt   i;
UMediumInt   os;
```

Public Function

## Purpose

The MemCopy function copies a region of the array pMem to another region. The destination index is di, the source index is i, the size of the array in elements is s, the number of elements to copy is n, and the size of each element is os.

Parameter	-	Description
-----------	---	-------------

---

pMem	-	Pointer to <i>Memory</i> .
s	-	Size of pMem in elements.
di	-	Destination index.
n	-	Number of elements to copy.
i	-	Source index.
os	-	Size of each element.

## Return Value

No return value

## Notes

[i must be less s.]

[(i + n) must be less or equal to s.]

[os must be greater than 0.]

[di must be less s.]

# MemClear

---

## Summary

```
#include "cobjects.h"
#include "memmac.h"
```

```
Void      MemClear( pMem, s, i, n, os )
Mem      *pMem;
UMediumInt s;
UMediumInt i;
UMediumInt n;
UMediumInt os;
```

Public Function

## Purpose

The MemClear function clears a section of the array pMem to zeros. The size of the array in elements is s, the beginning index is i, the number of elements to clear is n, and the size of each element is os.

## Parameter - Description

---

pMem	-	Pointer to <i>Memory</i> .
s	-	Size of pMem in elements.
i	-	First array index to start with.
n	-	Number of elements to clear.
os	-	Size of each element.

## Return Value

No return value

## Notes

[i must be less s.]

[(i + n) must be less or equal to s.]

[os must be greater than 0.]

## Class Reference for *Memory*

<b>Structure Name:</b>	Memory
<b>Abbreviation:</b>	Mem
<b>Class Type:</b>	Primitive Class

## ***MetaClass***

---

This page is intentionally left blank

## Summary

```
#include "cobjects.h"  
#include "mclmac.h"
```

```
Void      MclValidate( pMcl )  
PMCL      pMcl;
```

Public Function

## Purpose

The MclValidate function verifies that the *MetaClass* pMcl is a valid instance.

Parameter	-	Description
-----------	---	-------------

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
------	---	---

## Return Value

No return value

## Notes

Because instances of type *MetaClass* are typically statically defined and initialized at compile time, this function provides a way of verifying that an instance was initialized properly.

Part of the validation of the instance is calling MclValidateMessages and MclValidateSuperClasses.

An invalid instance will generate an exception.

## See Also

MclValidateMessages, MclValidateSuperClasses

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclValidate function.



# MclSendDestroyClass

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
Void      MclSendDestroyClass( pMcl, pCls )
PMCL      pMcl;
PCLS      pCls;
```

Public Function

## Purpose

The MclSendDestroyClass function deallocates an instance of the *Class* pCls as specified by the *MetaClass* pMcl.

## Parameter - Description

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
------	---	---

## Return Value

No return value

## See Also

MclCreateClass, MclDestroyClass, MclSendCreateClass

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclSendDestroyClass function.

# MclSendCreateClass

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
PCLS      MclSendCreateClass( pMcl )
PMCL      pMcl;
```

Public Function

## Purpose

The MclSendCreateClass function returns an instance of the *Class* described by the *MetaClass* pMcl.

## Parameter - Description

---

pMcl - Pointer to a structure of type *MetaClass*.

## Return Value

The return value from the MclSendCreateClass function is an instance of the *Class* described by the *MetaClass* pMcl.

## See Also

MclCreateClass, MclDestroy, MclSendDestroyClass

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclSendCreateClass function.

# MclPrint

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
Void          MclPrint( pMcl, pCio, item, level, name )
PMCL          pMcl;
PCIO          pCio;
MediumInt     item;
MediumInt     level;
PSTR          name;
```

Public Function

## Purpose

The MclPrint function prints the contents of the *MetaClass* pMcl on the ConsoleInputOutput device pCio. The item parameter is the array index of this instance or -1 if it is not an array element, level is a number indicating the level of indentation, and name is a *String* pointer which is the name of this instance.

## Parameter - Description

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
pCio	-	Pointer to a structure of type ConsoleInputOutput.
item	-	The array element number of this instance (or -1).
level	-	the level of indentation to print this object with.
name	-	the name of this instance.

## Return Value

No return value

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclPrint function.

# MclGetSuperClassCount

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
MediumInt    MclGetSuperClassCount( pMcl )
PMCL         pMcl;
```

Public Function

## Purpose

The MclGetSuperClassCount function returns the number of superclasses to this *MetaClass*.

Parameter	-	Description
-----------	---	-------------

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
------	---	---

## Return Value

The return value from the MclGetSuperClassCount function is the number of superclasses in this *MetaClass*.

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclGetSuperClassCount function.

# MclGetNthSuper

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
PMCL      MclGetNthSuper( pMcl, n )
PMCL      pMcl;
MediumInt n;
```

Public Function

## Purpose

The MclGetNthSuper function returns the Nth superclass *MetaClass* of the *MetaClass* pMcl.

Parameter	-	Description
-----------	---	-------------

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
n	-	Index of the superclass being returned.

## Return Value

The return value from the MclGetNthSuper function is the index in the list of MetaMessages understood by pMcl. If the selector is not found, -1 is returned.

## Notes

[The super class index must be greater than or equal to zero, and less than the number of superclasses.]

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclGetNthSuper function.

# MclGetNthOffset

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
MediumInt    MclGetNthOffset( pMcl, n )
PMCL         pMcl;
MediumInt    n;
```

Public Function

## Purpose

The MclGetNthOffset function returns the offset of the Nth superclass of the *MetaClass* pMcl. The offset is used to calculate the distance between an object instance and its superclass.

## Parameter - Description

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> . The <i>MetaClass</i> being searched.
n	-	Index of the superclass whose offset is being calculated.

## Return Value

The return value from the MclGetNthOffset function is the offset of the Nth superclass from its subclass as specified by the *MetaClass* pMcl.

## Notes

For the offset to be calculated correctly, the example instance of an object of this *MetaClass* type must be the same object instance referenced by the *MetaSuperClass* described by this *MetaClass*.

[The super class index must be greater than or equal to zero, and less than the number of superclasses.]

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclGetNthOffset function.

# MclGetNthMms

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
PMMS      MclGetNthMms( pMcl, n )
PMCL      pMcl;
MediumInt n;
```

Public Function

## Purpose

The MclGetNthMms function returns the Nth *MetaMessage* described by the *MetaClass* pMcl.

Parameter	-	Description
-----------	---	-------------

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
n	-	The index of the <i>MetaMessage</i> to return.

## Return Value

The return value from the MclGetNthMms function is a pointer to a structure of type *MetaMessage* which is the Nth *MetaMessage* described by this *MetaClass*.

## Notes

[The message number must be greater than or equal to zero, and less than the number of messages.]

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclGetNthMms function.

# MclGetMessageCount

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
MediumInt    MclGetMessageCount( pMcl )
PMCL         pMcl;
```

Public Function

## Purpose

The MclGetMessageCount function returns the total number of MetaMessages which are included as part of the *MetaClass* instance pMcl.

## Parameter - Description

---

pMcl - Pointer to a structure of type *MetaClass*.

## Return Value

The return value from the MclGetMessageCount function is the number of messages responded to by an instance of this type.

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclGetMessageCount function.



# MclGetClassSize

---

## Summary

```
#include "cobjects.h"  
#include "mclmac.h"
```

```
UMediumInt  MclGetClassSize( pMcl )  
PMCL        pMcl;
```

Public Function

## Purpose

The MclGetClassSize function returns the size of an instance of the *Class* described by the *MetaClass* pMcl.

## Parameter - Description

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> . The <i>MetaClass</i> being searched.
------	---	--

## Return Value

The return value from the MclGetClassSize function returns the size of an instance of the *MetaClass* pMcl.

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclGetClassSize function.

# MclGetClassName

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
PSTR      MclGetClassName( pMcl )
PMCL      pMcl;
```

Public Function

## Purpose

The MclGetClassName function returns the name of the class described by the *MetaClass* pMcl.

## Parameter - Description

---

pMcl - Pointer to a structure of type *MetaClass*.

## Return Value

The return value from the MclGetClassName function is the name of the *Class* described by the *MetaClass*.

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclGetClassName function.

# MclFindSuperClass

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
MediumInt    MclFindSuperClass( pMcl, pStr )
PMCL          pMcl;
PSTR          pStr;
```

Public Function

## Purpose

The MclFindSuperClass function looks for the superclass pStr in the *MetaClass* pMcl and returns its index which can be used to locate its *MetaClass*. The function returns -1 if the superclass is not located.

## Parameter - Description

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> . The <i>MetaClass</i> being searched.
pStr	-	Pointer to a <i>String</i> . The superclass name to search for.

## Return Value

The return value from the MclFindSuperClass function is the index in the list of *MetaSuperClass* understood by pMcl. If the superclass is not found, -1 is returned.

## Notes

The comparison between pStr and the superclass name is case sensitive.

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclFindSuperClass function.

# MclFindSelector

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
MediumInt    MclFindSelector( pMcl, pStr )
PMCL         pMcl;
PSTR         pStr;
```

Public Function

## Purpose

The MclFindSelector function looks for the selector pStr in the *MetaClass* pMcl and returns its index which can be used to invoke the message. The function returns -1 if the selector is not located.

## Parameter - Description

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> . The <i>MetaClass</i> being searched.
pStr	-	Pointer to a <i>String</i> . The selector name to search for.

## Return Value

The return value from the MclFindSelector function is the index in the list of MetaMessages understood by pMcl. If the selector is not found, -1 is returned.

## Notes

The comparison between pStr and the selector name is case sensitive.

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclFindSelector function.

# MclDestroyClass

---

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclDestroyClass function.

# MclDestroyClass

---

## Summary

```
#include "cobjects.h"
#include "mclmac.h"
```

```
Void      MclDestroyClass( pMcl, pCls )
PMCL      pMcl;
PCLS      pCls;
```

Friend Function

## Purpose

The MclDestroyClass function deallocates the class instance pCls previously allocated by the *MetaClass* pMcl.

Parameter	-	Description
-----------	---	-------------

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
pCls	-	Pointer to a structure of type <i>Class</i> . The class instance to be deallocated.

## Return Value

No return value

## Notes

pCls must have been previously allocated by MclSendCreateClass using pMcl.

Rather than using this function directly, the equivalent function SendDestroyClass should be called. This makes sure that if pCls is not the root class, that it is deinitialized properly before being deallocated.

*Class* calls this function to deallocate the memory for the class after it has been deinitialized.

## See Also

MclCreateClass, MclSendCreateClass, MclSendDestroyClass

# MclCreateClass

---

## Summary

```
#include "cobjects.h"  
#include "mclmac.h"
```

```
PCLS      MclCreateClass( pMcl )  
PMCL      pMcl;
```

Private Function

## Purpose

The MclCreateClass function creates a new class instance based on the *MetaClass* pMcl.

Parameter	-	Description
-----------	---	-------------

---

pMcl	-	Pointer to a structure of type <i>MetaClass</i> .
------	---	---

## Return Value

The return value from the MclCreateClass function is a pointer to a structure of type *Class* and is the new class instance.

## Notes

Rather than using CreateClass directly, a user should use the equivalent SendCreateClass function. This guarantees that if pMcl describes an extended *Class* that it is initialized properly.

## See Also

MclDestroyClass, MclSendCreateClass, MclSendDestroyClass

## Example

Please refer to class test procedure TSTMCL.C for an example of the use of the MclCreateClass function.

## Class Reference for *MetaClass*

<b>Structure Name:</b>	MetaClass
<b>Abbreviation:</b>	Mcl
<b>Class Type:</b>	Primitive Class



## ***ListElement***

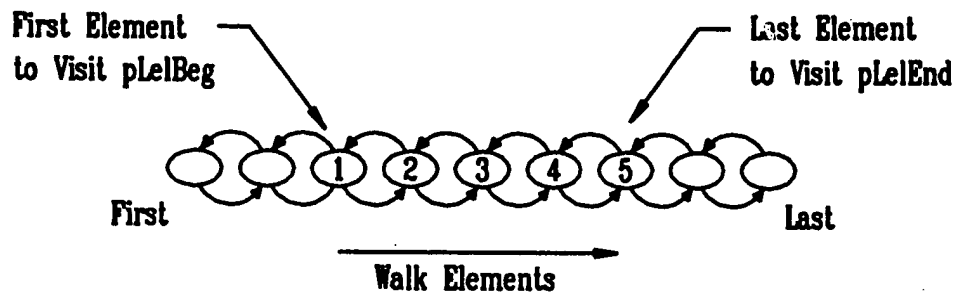
---

This page is intentionally left blank

# LeIVisitRange

---

## Diagram



# LelVisitRange

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void      LelVisitRange( pLelBeg, pLelEnd, pBlk )
PLEL      pLelBeg;
PLEL      pLelEnd;
PBLK      pBlk;
```

Private Function

## Purpose

The LelVisitRange function walks the list for the range of *ListElements* pLelBeg through pLelEnd and calls a *ListElement* function for each list element visited.

The *Block* pBlk contains the function and an optional list of arguments.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *ListElement* class section on range definition for more details.

## Parameter - Description

---

pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the beginning list element in a range.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the ending list element in a range.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

The *ListElement* function may return a value but it is ignored.

pLelBeg and pLelEnd must belong to the same list.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void      LelTest( pLel )
PLEL      pLel;
```

Public Function

## Purpose

The LelTest function validates the *ListElement* pLel. An invalid *ListElement* will generate an exception.

## Return Value

No return value

## Notes

[The successor list element must point to pLel.]

[The predecessor list element must point to pLel.]

## Example

Please refer to class test procedure TSTLEL.C,TSTDLL.C for an example of the use of the LelTest function.

# LelSendDestroy

---

## Summary

```
#include "cobjects.h"  
#include "lelmac.h"
```

```
Void      LelSendDestroy( pLel )  
PLEL      pLel;
```

Public Function

## Purpose

The LelSendDestroy function sends a message to the client of the *ListElement* pLel asking it to destroy pLel. The *ListElement* client function will receive this message and should destroy the list element. This message function should be included in the *ListElement* client message array.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
------	---	---

## Return Value

No return value

## Example

Please refer to class test procedure TSTLEL.C,TSTDLL.C for an example of the use of the LelSendDestroy function.

## LelPasteRangeBefore

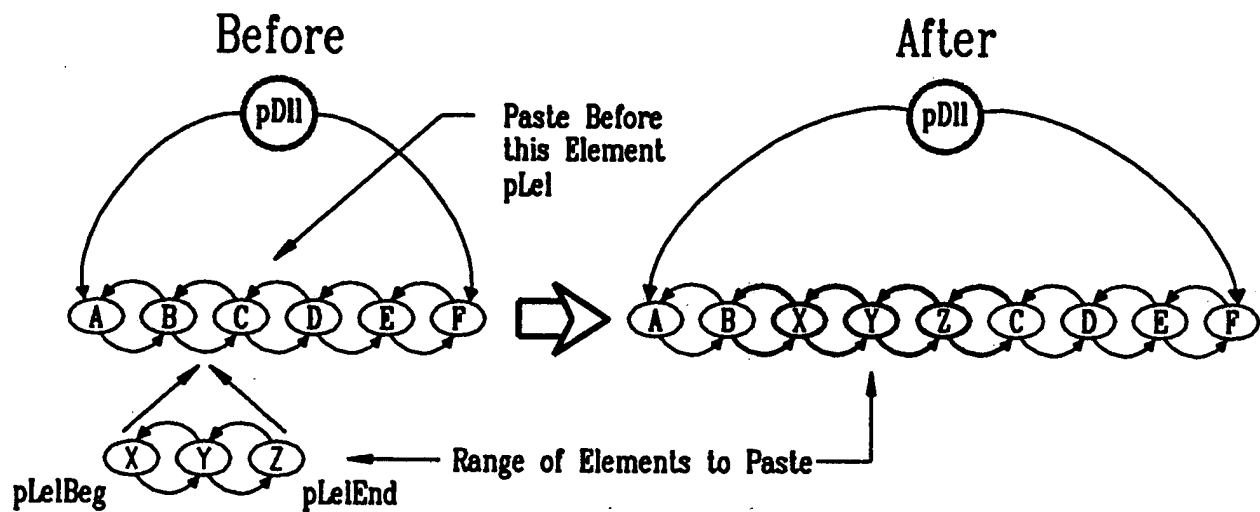
### Notes (cont)

[pLel cannot be identical to either pLelBeg or pLelEnd.]

### See Also

LelCut, LelCutRange, LelPasteRangeAfter

### Diagram



# LelPasteRangeBefore

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void      LelPasteRangeBefore( pLel, pLelBeg, pLelEnd )
PLEL      pLel;
PLEL      pLelBeg;
PLEL      pLelEnd;
```

Public Function

## Purpose

The LelPasteRangeBefore function links a range of *ListElements* pLelBeg through pLelEnd to the *List* containing pLel, preceding the *ListElement* pLel.

The list element range must have been previously cut from a list.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *ListElement* class section on range definition for more details.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> . The elements will be linked preceding this list element. It cannot be NULL.
pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to link to the list.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to link to the list.

## Return Value

No return value

## Notes

pLelBeg and pLelEnd must belong to same list if any.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

[pLelBeg must not already be in a list.]

## LelPasteRangeAfter

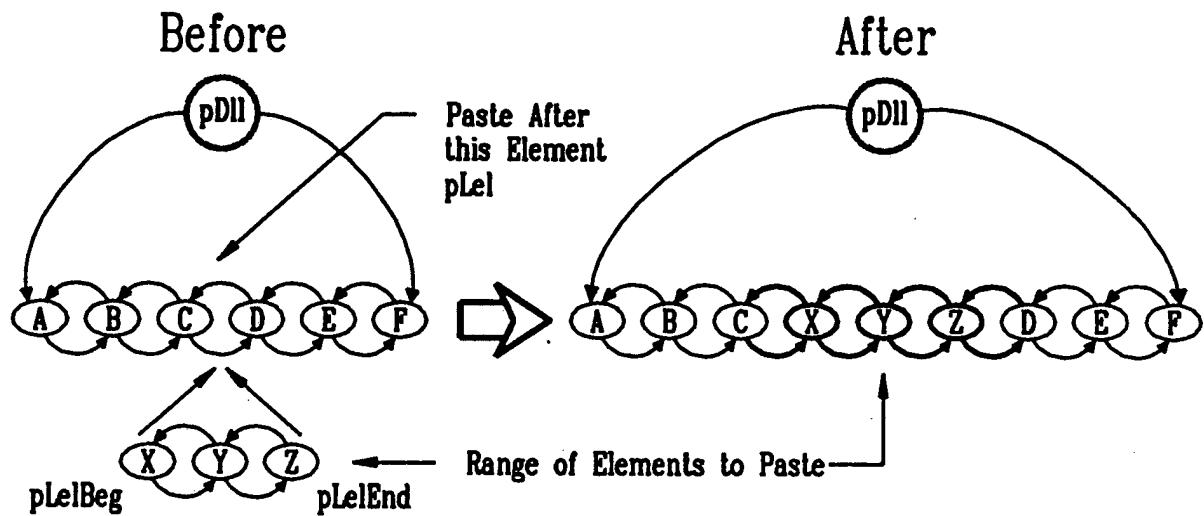
### Notes (cont)

[pLel cannot be identical to either pLelBeg or pLelEnd.]

### See Also

LelCut, LelCutRange, LelPasteRangeBefore

### Diagram





# LelPasteRangeAfter

---

## Summary

```
#include "cobjects.h"
```

```
#include "lelmac.h"
```

```
Void      LelPasteRangeAfter( pLel, pLelBeg, pLelEnd )
PLEL      pLel;
PLEL      pLelBeg;
PLEL      pLelEnd;
```

Public Function

## Purpose

The LelPasteRangeAfter function links a range of *ListElements* pLelBeg through pLelEnd to the *List* containing pLel, succeeding the *ListElement* pLel.

The list element range must have been previously cut from a list.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *ListElement* class section on range definition for more details.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> . The elements will be linked succeeding this list element. It cannot be NULL.
pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to paste into in a list.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to paste into in a list.

## Return Value

No return value

## Notes

pLelBeg and pLelEnd must belong to the same list if any.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

[pLelBeg must not already be in a list.]

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void      LelInit( pLel )
PLEL      pLel;
```

Public Function

## Purpose

The LelInList function determines whether the *ListElement* pLel is in a list.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
------	---	---

## Return Value

No return value

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelInit function.

# LelGetPrev

---

## Summary

```
#include "cobjects.h"  
#include "lelmac.h"
```

```
PLEL      LelGetPrev( pLel )  
PLEL      pLel;
```

Friend Function

A macro is available for this function

## Purpose

The LelGetPrev function returns a pointer to the preceding list element to the *ListElement* pLel.

## Parameter - Description

---

pLel - Pointer to a structure of type *ListElement*.

## Return Value

The return value from the LelGetPrev function is a pointer to the list element preceding the *ListElement* pLel.

## See Also

LelGetNext, LelGetNthSuccessor

## Example

Please refer to class test procedure TSTLEL.C,TSTDLL.C for an example of the use of the LelGetPrev function.

# LelGetNthSuccessor

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
PLEL      LelGetNthSuccessor( pLel, offset )
PLEL      pLel;
MediumInt offset;
```

Friend Function

## Purpose

The LelGetNthSuccessor function returns a pointer to the list element which is the Nth successor element forward in the list from the *ListElement* pLel.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
offset	-	Offset from the <i>ListElement</i> pLel to another list element. [0-END]

offset: END is the last consecutively numbered list element index.

## Return Value

The return value from the LelGetNthSuccessor function is a pointer to the Nth successor list element after the *ListElement* pLel.

## See Also

LelGetNext, LelGetNthSuccessor, LelGetPrev

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelGetNthSuccessor function.

# LelGetNext

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
PLEL      LelGetNext( pLel )
PLEL      pLel;
```

## Friend Function

A macro is available for this function

## Purpose

The LelGetNext function returns a pointer to the succeeding list element to the *ListElement* pLel.

## Parameter - Description

---

pLel - Pointer to a structure of type *ListElement*.

## Return Value

The return value from the LelGetNext function is the succeeding list element to the *ListElement* pLel.

## See Also

LelGetNthSuccessor, LelGetPrev

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelGetNext function.

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
PDLL      LelGetDll( pLel )
PLEL      pLel;
```

Friend Function

A macro is available for this function

## Purpose

The LelGetDll function returns a pointer to the *List* that the *ListElement* pLel is linked to.

Parameter	-	Description
-----------	---	-------------

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
------	---	---

## Return Value

The return value from the LelGetDll function is a pointer to the *List* that the *ListElement* pLel is linked to. If the *ListElement* pLel does not belong to a list then NULL is returned.

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelGetDll function.

# LelGetClient

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
POBJ      LelGetClient( pLel, offset )
PLEL      pLel;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The LelGetClient function returns the client pointer of the *ListElement* pLel.

## Parameter - Description

---

pLel - Pointer to a structure of type *ListElement*.

## Return Value

The return value from the LelGetClient function is the client pointer of the *ListElement* pLel.

## See Also

LelClientNext, LelClientPrev

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelGetClient function.

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void      LelDestroy( pLel )
PLEL      pLel;
```

Public Function

## Purpose

The LelDestroy function deallocates the memory used by the object and deinitializes the *ListElement* object. The *ListElement* pLel should not be referenced after this function call since its memory will have been deallocated.

The list element is cut from the list prior to before destroying the object.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

No return value

## Notes

[The instance must be the outermost subclass; it cannot have a sub-object.]

## See Also

LelDeInit, LelInit

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelDestroy function.



# LelDeInit

---

## Summary

```
#include "cobjects.h"  
#include "lelmac.h"
```

```
Void      LelDeInit( pLel )  
PLEL      pLel;
```

Public Function

## Purpose

The LelDeInit function deinitializes the *ListElement* object. The LelDeInit function should be the last function called when using the *ListElement* class.

## Parameter - Description

---

pLel - Pointer to a structure of type *ListElement*.

## Return Value

No return value

## Notes

The first function to call when using the *ListElement* class is LelInit.

pLel must be cut from the list prior to calling this function.

[The list element must not be connected to any other elements.]

[The list element must not be in a list.]

## See Also

LelDestroy, LelInit

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelDeInit function.

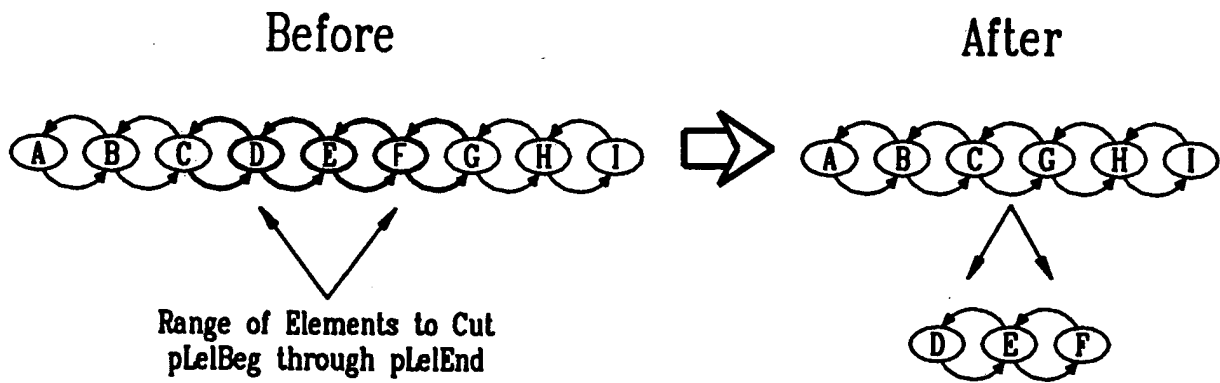
## LelCutRange

---

### See Also

LelCut, LelPasteRangeAfter, LelPasteRangeBefore

### Diagram



# LelCutRange

---

## Summary

```
#include "cobjects.h"
```

```
#include "lelmac.h"
```

```
Void          LelCutRange( pLelBeg, pLelEnd )  
PLEL          pLelBeg;  
PLEL          pLelEnd;  
{BLK
```

Public Function

## Purpose

The LelCutRange function unlinks the range of *ListElements* pLelBeg through pLelEnd from the list. The list element preceding pLelBeg will be linked to the successor list element of pLelEnd.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *ListElement* class section on range definition for more details.

## Parameter - Description

---

pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the beginning list element to cut.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the ending list element to cut.

## Return Value

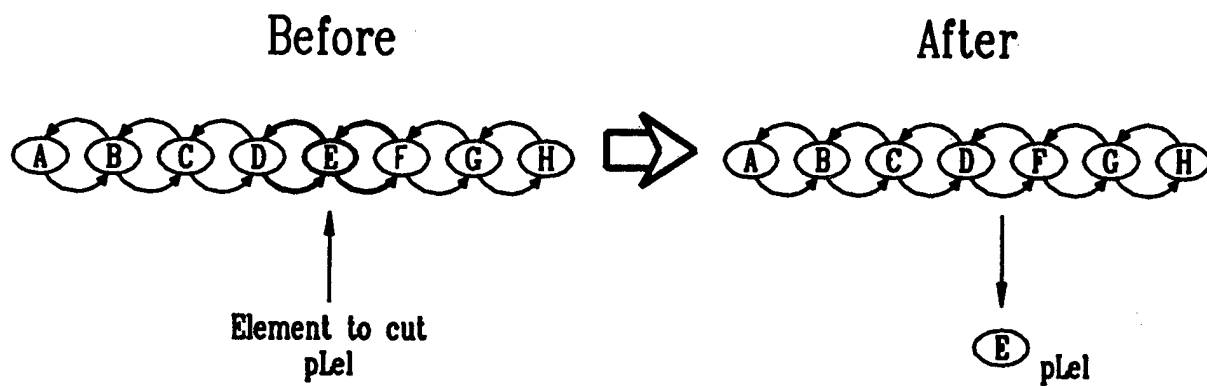
No return value

## Notes

pLelBeg and pLelEnd must belong to the same list.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

## Diagram



# LelCut

---

## Summary

```
#include "cobjects.h"  
#include "lelmac.h"
```

```
Void      LelCut( pLel )  
PLEL      pLel;
```

Public Function

## Purpose

The LelCut function unlinks the *ListElement* pLel from the list. The list element preceding pLel will be linked to the successor list element of pLel.

## Parameter - Description

---

pLel - Pointer to a structure of type *ListElement*.

## Return Value

No return value

## Notes

pLel must belong to a list.

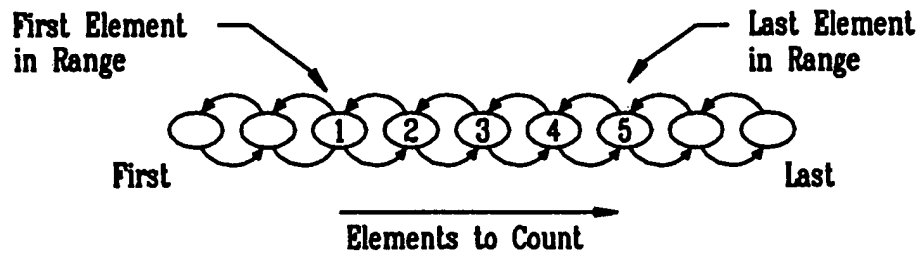
## See Also

LelCutRange, LelPasteRangeAfter, LelPasteRangeBefore

## LelCountRange

---

### Diagram



# LelCountRange

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
MediumInt    LelCountRange( pLelBeg, pLelEnd )
PLEL          pLelBeg;
PLEL          pLelEnd;
```

Public Function

## Purpose

The LelCountRange function returns the number of list elements in the range of *ListElements* pLelBeg through pLelEnd.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *ListElement* class section on range definition for more details.

## Parameter - Description

---

pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element in a range.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element in a range.

## Return Value

The return value from the LelCountRange function is the number of list elements in the range including the first and last *ListElements* pLelBeg and pLelEnd.

## Notes

pLelBeg and pLelEnd must belong to the same list.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

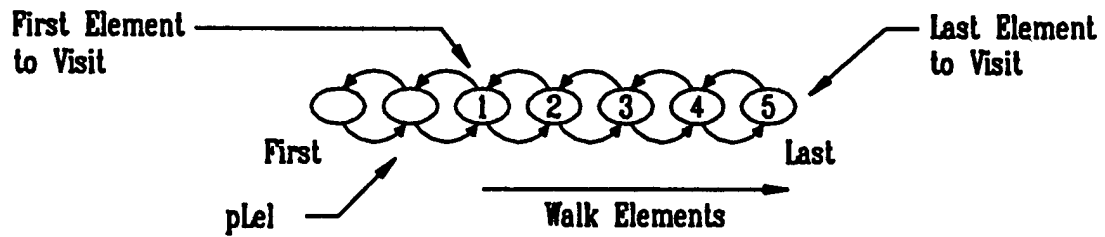
# LelClientVisitSuccessors

---

## See Also

LelClientVisitBwd, LelClientVisitFwd, LelClientVisitPredecessors,  
LelClientVisitRange

## Diagram





# LelClientVisitSuccessors

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void          LelClientVisitSuccessors( pLel, offset, pBlk )
PLEL          pLel;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The LelClientVisitSuccessors function walks the list in and calls a *ListElement* client function for each list element visited. The list is walked in a forward direction starting with the *ListElement* succeeding the *ListElement* pLel.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> . The successor to this element is the first visited.
offset	-	The distance in bytes between the <i>ListElement</i> pLel and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

## LelClientVisitRange

---

### Notes

The client function may return a value but it is ignored.

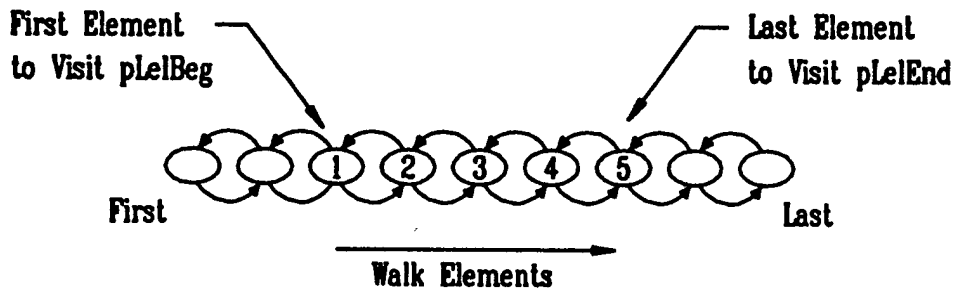
pLelBeg and pLelEnd must belong to the same list.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

### See Also

LelClientVisitBwd, LelClientVisitFwd, LelClientVisitPredecessors,  
LelClientVisitSuccessors

### Diagram



# LelClientVisitRange

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
MediumInt    LelClientVisitRange( pLelBeg, pLelEnd, offset, pBlk )
PLEL          pLelBeg;
PLEL          pLelEnd;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The `LelClientVisitRange` function walks the list for a range of *ListElements* `pLelBeg` through `pLelEnd` and calls a *ListElement* client function for each list element visited.

The *Block* `pBlk` contains the client function and an optional list of arguments.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *ListElement* class section on range definition for more details.

## Parameter - Description

---

<code>pLelBeg</code>	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to visit.
<code>pLelEnd</code>	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to visit.
<code>offset</code>	-	The distance in bytes between the <i>ListElement</i> <code>pLel</code> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the `LelClientVisitRange` function is the number of times list elements are visited.

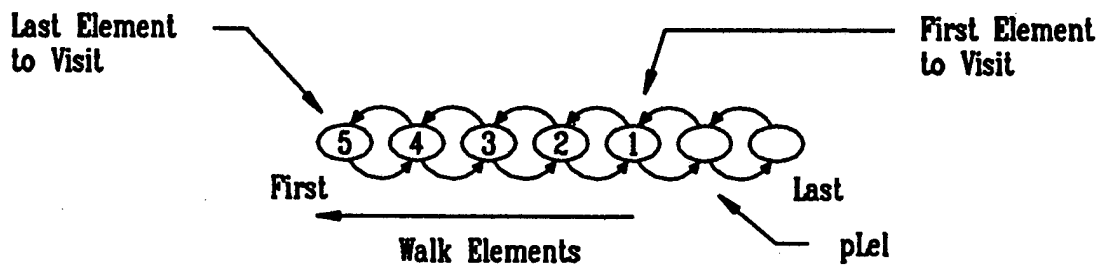
## LelClientVisitPredecessors

---

### See Also

LelClientVisitBwd, LelClientVisitFwd, LelClientVisitRange,  
LelClientVisitSuccessors

### Diagram



# LelClientVisitPredecessors

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void          LelClientVisitPredecessors( pLel, offset, pBlk )
PLEL          pLel;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The LelClientVisitPredecessors function walks the list and calls a *ListElement* client function for each list element visited. The list is walked in a backward direction starting with the *ListElement* preceding the *ListElement* pLel.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> . The predecessor to this element is the first visited.
offset	-	The distance in bytes between the <i>ListElement</i> pLel and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

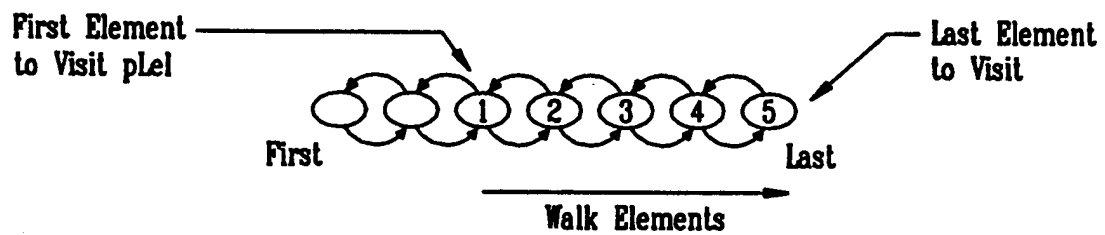
## LelClientVisitFwd

---

### See Also

LelClientVisitBwd, LelClientVisitPredecessors, LelClientVisitRange, LelClientVisitSuccessors

### Diagram



# LelClientVisitFwd

---

## Summary

```
#include "objects.h"
#include "lelmac.h"
```

```
Void      LelClientVisitFwd( pLel, offset, pBlk )
PLEL      pLel;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The `LelClientVisitFwd` function walks the list and calls a *ListElement* client function for each list element visited. The list is walked in a forward direction (self to last) starting with the *ListElement* `pLel`.

The *Block* `pBlk` contains the client function and an optional list of arguments.

## Parameter - Description

---

<code>pLel</code>	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element visited.
<code>offset</code>	-	The distance in bytes between the <i>ListElement</i> <code>pLel</code> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

[The list element may be NULL in which case no elements are visited.]

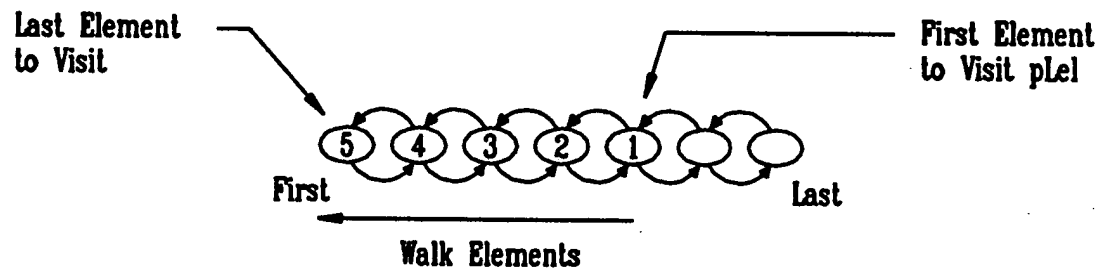
## LelClientVisitBwd

---

### See Also

LelClientVisitFwd, LelClientVisitPredecessors, LelClientVisitRange,  
LelClientVisitSuccessors

### Diagram





# LelClientVisitBwd

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
Void          LelClientVisitBwd( pLel, offset, pBlk )
PLEL          pLel;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The LelClientVisitBwd function walks the list and calls a *ListElement* client function for each list element visited. The list is walked in a backward direction (self to first) starting with the *ListElement* pLel.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element visited.
offset	-	The distance in bytes between the <i>ListElement</i> pLel and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

[The list element may be NULL in which case no elements are visited.]

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
POBJ      LelClientPrev( pLel, offset )
PLEL      pLel;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The LelClientPrev function returns the client pointer of the preceding list element to the *ListElement* pLel.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
offset	-	The distance in bytes between the <i>ListElement</i> pLel and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the LelClientPrev function is the client pointer of the preceding list element to the *ListElement* pLel or NULL if pLel is the first element.

## See Also

LelGetClient, LelClientNext

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelClientPrev function.

# LelClientNext

---

## Summary

```
#include "cobjects.h"
```

```
#include "lelmac.h"
```

```
POBJ      LelClientNext( pLel, offset )
PLEL      pLel;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The LelClientNext function returns the client pointer of the succeeding list element to the *ListElement* pLel.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
offset	-	The distance in bytes between the <i>ListElement</i> pLel and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the LelClientNext function is the client pointer of the succeeding list element to the *ListElement* pLel or NULL if pLel is the last element.

## See Also

LelGetClient, LelClientPrev

## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelClientNext function.

## LelClientFindRange

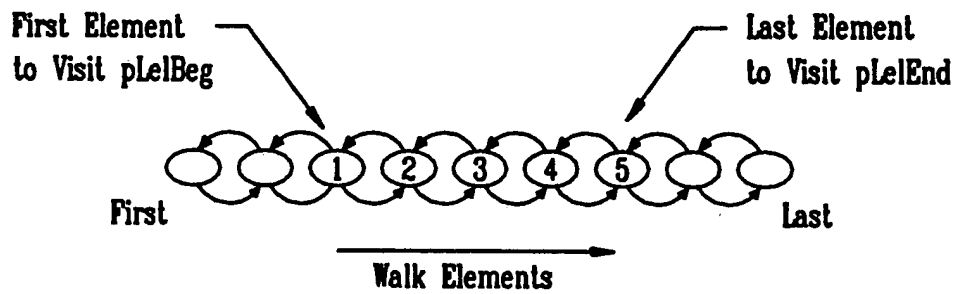
---

### Notes

pLelBeg and pLelEnd must belong to the same list.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

### Diagram



# LelClientFindRange

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
POBJ      LelClientFindRange( pLelBeg, pLelEnd, offset, pBlk )
PLEL      pLelBeg;
PLEL      pLelEnd;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The LelClientFindRange function walks the list for the range of *ListElements* pLelBeg through pLelEnd and calls a client function for each list element visited. The list is walked in a forward direction. The function terminates when the client function returns True or the end of the list is reached. If a True value is returned, the client pointer of the list element is returned, otherwise NULL is returned.

The *Block* pBlk contains the client function and an optional list of arguments. The function must return a boolean (True/False) value.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *ListElement* class section on range definition for more details.

## Parameter - Description

---

pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to visit.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to visit.
offset	-	The distance in bytes between the <i>ListElement</i> pLel and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the LelClientFindRange function is the client pointer of the first list element for which the client function returns True. Otherwise NULL is returned.

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
POBJ      LelClientDll( pLel, offset )
PLEL      pLel;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The LelClientDll function returns the client pointer of the list that the *ListElement* pLel is linked to.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
------	---	---

## Return Value

The return value from the LelClientDll function is the client pointer of the list that the *ListElement* pLel is linked to or NULL if it is not linked to a list.

## See Also

LelGetDll

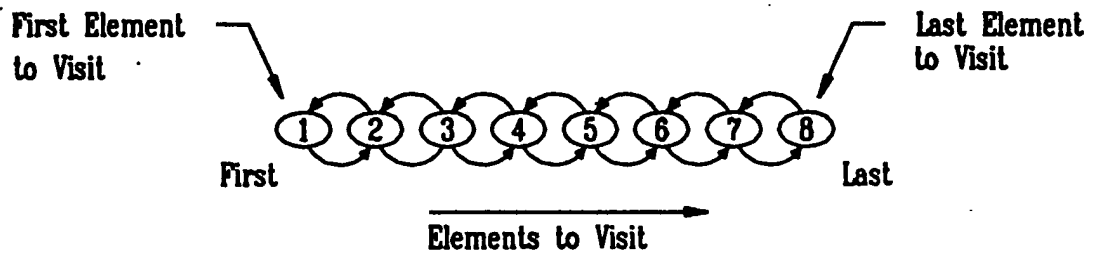
## Example

Please refer to class test procedure TSTLEL.C, TSTDLL.C for an example of the use of the LelClientDll function.

# LelClientCount

---

## Diagram



# LelClientCount

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
MediumInt    LelClientCount( pLel, offset, pBlk )
PLEL          pLel;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The LelClientCount function returns the number of times a *ListElement* client function returns non-zero. The entire list is walked in a forward direction starting with the *ListElement* pLel.

The *Block* pBlk contains the client function and an optional list of arguments. The function must return a MediumInt value.

Parameter	-	Description
-----------	---	-------------

---

pLel	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element visited.
offset	-	The distance in bytes between the <i>ListElement</i> pLel and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the LelClientCount function is the number of times the *ListElement* client function returns non-zero.

## Notes

[The list element can be NULL in which case the return value is zero and no items are processed.]



# LelAsObj

---

## Summary

```
#include "cobjects.h"
#include "lelmac.h"
```

```
POBJ      LelAsObj( pLel )
PLEL      pLel;
```

Private Function

A macro is available for this function

## Purpose

The LelAsObj function returns a pointer to the *Object* structure contained by the *ListElement* pLel.

## Parameter - Description

---

pLel	-	Pointer to a structure of type <i>ListElement</i> .
------	---	---

## Return Value

The return value from the LelAsObj function is a pointer to the *Object* structure contained by the *ListElement* class.

## Notes

The *Object* pointer can be used to send a message to the client of pLel.

## Example

Please refer to class test procedure TSTLEL.C,TSTDLL.C for an example of the use of the LelAsObj function.

## Class Reference for *ListElement*

<b>Structure Name:</b>	ListElement
<b>Abbreviation:</b>	Lel
<b>Class Type:</b>	Inheritable class

## ***Julian Time***

---

This page is intentionally left blank

## JulYearString

---

### Example

```
{
Jul jul;
Char prtBuf[5];
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* get the year string */
JulYearString( &jul, prtBuf );
printf( "This year is %s\n", prtBuf );
}
```

The following output will appear to the terminal  
This year is 1987

# JulYearString

---

## Summary

#include "cobjects.h"

#include "julmac.h"

Void                   JulYearString( pJul, pStr )  
PJUL                   pJul;  
PSTR                   pStr;

Public Function

## Purpose

The JulYearString function returns the *String* pStr filled with the year represented by *JulianTime* pJul. The format is:

example: 1987   or   1988  
          4 characters

## Parameter   -   Description

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
pStr	-	Returns the year string. The string should be able to hold 4 characters plus a NULL terminator.

## Return Value

No return value

## Notes

This function is used to create string output based upon date values. The output format can be controlled to any convention by editing the *Class Source* file jul.c and examining the sprintf statement within the function.

## See Also

JulWeekString, JulMonthString, JulQuarterString

## JulWeekString

---

### Example

```
{
Jul jul;
Char prtBuf[7];
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* get the week string */
JulWeekString( &jul, prtBuf );
printf( "This is the %s\n", prtBuf );
.
}
```

The following output will appear to the terminal  
This is the 23 Dec

# JulWeekString

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulWeekString( pJul, pStr )
PJUL      pJul;
PSTR      pStr;
```

Public Function

## Purpose

The `JulWeekString` function returns the *String* `pStr` filled with the day and month represented by *JulianTime* `pJul`. The format is:

example: 2 Aug    or    18 Jul  
          1-2 digits left justified + 3 characters

---

Parameter	-	Description
-----------	---	-------------

---

<code>pJul</code>	-	Pointer to a structure of type <i>JulianTime</i> .
<code>pStr</code>	-	Returns the week string. The string should be able to hold 6 characters plus a NULL terminator.

## Return Value

No return value

## Notes

This function is used to create string output based on julian time values. The output format can be controlled to any convention by editing the *Class Header* file `julmac.h` and replacing the `weekMonthName ClassData` string values with any requirement. The output format can further be controlled by editing the *Class Source* file `jul.c` and examining the `sprintf` statement within the function.

## See Also

`JulMonthString`, `JulQuarterString`, `JulYearString`

## JulValidateDate

---

### See Also

JulCalendarToJulian, JulDateStrToJulian, JulToDateStr

### Example

```
{
    Jul jul;
    Char dateBuf[11];
    .
    .
    /* get string from user */
    printf( "Enter date:" );
    gets( dateBuf );

    /* validate a julian date */
    if( JulValidateDate( dateBuf ) == True )
        printf( "Good date\n" );
    else
        printf( "Bad date\n" );
}
Example of a bad date is:
30-2-1987
```



# JulValidateDate

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Bool      JulValidateDate( pStr, format )
PSTR      pStr;
DateFormat format;
```

Public Function

## Purpose

The JulValidateDate function will validate a date represented by the *String* pStr. The format of the string is controlled by a date format identifier.

This function is not typical of a class function in that a *JulianTime* pointer is not the first argument.

## Parameter - Description

---

pStr	-	<i>String</i> containing the date.
format	-	Date string format.

date format:

```
DF_US      > MM-DD-YYYY
DF_EUROPE  > DD-MM-YYYY
DF_MILITARY > YYYY-MM-DD
```

## Return Value

The return value from the JulValidateDate function is True if the date is valid and False if the date is invalid.

## Notes

This function is used to validate date strings prior to converting them to julian time.

### Example

```
{
    Jul jul;
    Char dateBuf[11];
    .
    .
    /* create a julian date */
    JulDateStrToJulian( &jul, "12-23-1987", DF_US );

    /* convert to a date string */
    JulToDateStr( &jul, dateBuf, DF_US );
    printf( "This is the date %s\n", dateBuf );
}
```

The following output will appear to the terminal  
This is the date 12-23-1987

# JulToDateStr

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulToDateStr( pJul, pStr, format )
PJUL      pJul;
PSTR      pStr;
DateFormat format;
```

Public Function

## Purpose

The JulToDateStr function returns the *String* pStr filled with the date represented by *JulianTime* pJul. The format of the date string is controlled by passing a date format identifier.

## Parameter - Description

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
pStr	-	Returns a date string. The string should be able to hold 10 characters plus a NULL terminator.
format	-	Date string format.

date format:

```
DF_US      > MM-DD-YYYY
DF_EUROPE  > DD-MM-YYYY
DF_MILITARY > YYYY-MM-DD
```

## Return Value

No return value

## See Also

JulDateStrToJulian, JulValidateDate

## JulToCalendar

---

### Example

```
{
    Jul    jul;
    MediumInt year;
    MediumInt month;
    MediumInt day;
    .
    .
    /* create the JulianTime pointer */
    JulCalendarToJulian( &jul, 1987, 1, 31 );

    /* convert the same date back */
    JulToCalendar( &jul, &year, &month, &day );
    printf( "date %d-%d-%d\n", month, day, year );
}
```

The following output will appear to the terminal:  
date 1-31-1987

# JulToCalendar

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulToCalendar( pJul, year, month, day )
PJUL          pJul;
MediumInt     *year;
MediumInt     *month;
MediumInt     *day;
```

Public Function

## Purpose

The JulToCalendar function returns the values of year, month, and day, for the *JulianTime* pJul.

## Parameter - Description

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> year.
year	-	Returns a 4 digit year. [1583 - 4713]
month	-	Returns a 1-2 digit month. [1 - 12]
day	-	Returns a 1-2 digit day. [1 - 31]

## Return Value

No return value

## See Also

JulCalendarToJulian

### Example

```
{
    Jul    jul;
    .
    .
    JulSetMaxDate( &jul ); /* set date to maximum */

    /* check if date is maximum value */
    if( JulIsMaxValue( &jul ) == True )
        printf( "date is maximum\n" );
    else
        printf( "date is NOT maxium\n" );
}
```

The following output will appear to the terminal:  
date is maximum

# JulSetMaxDate

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulSetMaxDate( pJul )
PJUL      pJul;
```

Public Function

## Purpose

The JulSetMaxDate function places the maximum julian time value into *JulianTime* pJul.

## Parameter - Description

---

pJul - Pointer to a structure of type *JulianTime*.

## Return Value

No return value

## Notes

Use the JullIsMaxValue function to compare against the maximum julian time value.

The value assigned to pJul is julian day number 2147483647.

## See Also

JullIsMaxValue

## JulSameDayMonth

---

### Example

```
{
Jul jul1;
Jul jul2;

/* create two julian dates */
JulDateStrToJulian( &jul1, "12-23-1987", DF_US );
JulDateStrToJulian( &jul2, "12-23-1934", DF_US );

/* see if the date is the same within a year */
if( JulSameDayMonth( &jul1, &jul2 ) == True )
    printf( "Same day and month\n" );
else
    printf( "NOT the same day and month\n" );
}
```

The following output will appear to the terminal:  
Same day and month



# JulSameDayMonth

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Bool      JulSameDayMonth( pJul1, pJul2 )
PJUL      pJul1;
PJUL      pJul2;
```

Public Function

## Purpose

The JulSameDayMonth function returns True if *JulianTime* pJul1 and *JulianTime* pJul2 have the same day and month values. The year value can be different.

## Parameter - Description

---

pJul1	-	Pointer to a structure of type <i>JulianTime</i> .
pJul2	-	Pointer to a structure of type <i>JulianTime</i> .

## Return Value

The return value from the JulSameDayMonth function is True if pJul1 and pJul2 have the same day and month values. The year value can be different.

## JulQuarterString

---

### Example

```
{
    Jul jul;
    Char prtBuf[13];
    .
    .
    /* create a julian date */
    JulDateStrToJulian( &jul, "12-23-1987", DF_US );

    /* get the quarter string */
    JulQuarterString( &jul, prtBuf );
    printf( "This quarter is %s\n", prtBuf );
}
```

The following output will appear to the terminal  
This quarter is 4th Qtr 1987

# JulQuarterString

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulQuarterString( pJul, pStr )
PJUL          pJul;
PSTR          pStr;
```

Public Function

## Purpose

The `JulQuarterString` function returns the *String* `pStr` filled with the quarter and year represented by *JulianTime* `pJul`. The format is:

example: 1st Qtr 1987 or 4th Qtr 1988  
12 characters

---

Parameter	-	Description
-----------	---	-------------

---

<code>pJul</code>	-	Pointer to a structure of type <i>JulianTime</i> .
<code>pStr</code>	-	Returns the quarter string. The string should be able to hold 11 characters plus a NULL terminator.

## Return Value

No return value

## Notes

This function is used to create string output based on julian time values. The output format can be controlled to any convention by editing the *Class* header file `julmac.h` and replacing the `quarterName ClassData` string values with any requirement.

Calendar quarters are measured as a year divided into 4 periods of 3 months each (1st period Jan-Mar, 2nd period Apr-Jun, 3rd period Jul-Sep, 4th period Oct-Dec).

## See Also

`JulWeekString`, `JulMonthString`, `JulYearString`

## JulMonthString

---

### Example

```
{
    Jul jul;
    Char prtBuf[4];
    .
    .
    /* create a julian date */
    JulDateStrToJulian( &jul, "12-23-1987", DF_US );

    /* get the month string */
    JulMonthString( &jul, prtBuf );
    printf( "This month is %s\n", prtBuf );
}
```

The following output will appear to the terminal  
This month is Dec

# JulMonthString

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulMonthString( pJul, pStr )
PJUL          pJul;
PSTR          pStr;
```

Public Function

## Purpose

The JulMonthString function returns the *String* pStr filled with the month and year represented by *JulianTime* pJul. The format is:

example: Aug 1988    or    Jul 2052  
                    8 characters

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
pStr	-	Returns the month string. The string should be able to hold 8 characters plus a NULL terminator.

## Return Value

No return value

## Notes

This function is used to create string output based on julian time values. The output format can be controlled to any convention by editing the *Class* header file julmac.h and replacing the monthName ClassData string values with any requirement.

## See Also

JulWeekString, JulQuarterString, JulYearString

## JulMonthDayDiff

---

### Example

```
{
    Jul      jul;
    MediumInt diff;
    .
    .
    /* create a julian date */
    JulDateStrToJulian( &jul, "12-23-1987", DF_US );

    /* see if what the difference is */
    diff = JulMonthDayDiff( &jul, 12, 1 );
    printf( "Difference = %d\n", diff );
}
```

The following output will appear to the terminal:  
Difference = 23

# JulMonthDayDiff

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
MediumInt    JulMonthDayDiff( pJul, mnth, dy )
PJUL         pJul;
MediumInt    mnth;
MediumInt    dy;
```

Public Function

## Purpose

The JulMonthDayDiff function returns the number of days between (year of pJul)/mnth/dy and *JulianTime* pJul.

## Parameter - Description

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
mnth	-	A 1-2 digit month. [1 - 12]
dy	-	A 1-2 digit day. [1 - 31]

## Return Value

The return value from the JulMonthDayDiff function is; (year of pJul)/mnth/dy - (minus) pJul.

### Example

```
{
Jul jul;
Jul jul1;
Jul jul2;
Char dateBuf[11];
.
.
/* create two julian dates */
JulDateStrToJulian( &jul1, "12-23-1987", DF_US );
JulDateStrToJulian( &jul2, "10-5-1987", DF_US );

/* evaluate the minimum of the two dates */
JulMin( &jul, &jul1, &jul2 );

/* print the minimum date */
JulToDateStr( &jul, dateBuf, DF_US );
printf( "minimum date %s\n", dateBuf );
}
```

The following output will appear to the terminal:  
minimum date 10-5-1987



# JulMin

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulMin( pJul, pJul1, pJul2 )
PJUL      pJul;
PJUL      pJul1;
PJUL      pJul2;
```

Public Function

A macro is available for this function

## Purpose

The JulMin function places the minimum of *JulianTime* pJul1 and *JulianTime* pJul2 into *JulianTime* pJul.

## Parameter - Description

---

pJul	-	Returned minimum of two julian time values.
pJul1	-	Pointer to a structure of type <i>JulianTime</i> .
pJul2	-	Pointer to a structure of type <i>JulianTime</i> .

## Return Value

No return value

## See Also

JulMax

## Example

```
{
Jul jul;
Jul jul1;
Jul jul2;
Char dateBuf[11];
.
.
/* create two julian dates */
JulDateStrToJulian( &jul1, "12-23-1987", DF_US );
JulDateStrToJulian( &jul2, "10-5-1987", DF_US );

/* evaluate the maximum of the two dates */
JulMax( &jul, &jul1, &jul2 );

/* print the maximum date */
JulToDateStr( &jul, dateBuf, DF_US );
printf( "maximum date %s\n", dateBuf );
.
}
```

The following output will appear to the terminal:  
maximum date 12-23-1987

# JulMax

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulMax( pJul, pJul1, pJul2 )
PJUL      pJul;
PJUL      pJul1;
PJUL      pJul2;
```

Public Function

A macro is available for this function

## Purpose

The JulMax function places the maximum of *JulianTime* pJul1 and *JulianTime* pJul2 into *JulianTime* pJul.

## Parameter - Description

---

pJul	-	Returned maximum of two julian time values.
pJul1	-	Pointer to a structure of type <i>JulianTime</i> .
pJul2	-	Pointer to a structure of type <i>JulianTime</i> .

## Return Value

No return value

## Notes

pJul1 or pJul2 can be any julian time value.

## See Also

JulMin

## JullsMaxValue

---

### Example

```
{
Jul    jul;
.
.
JulSetMaxDate( &jul ); /* set date to maximum */

/* check if date is maximum value */
if( JulsMaxValue( &jul ) == True )
    printf( "date is maximum\n" );
else
    printf( "date is NOT maxium\n" );
}
```

The following output will appear to the terminal:  
date is maximum

# JullsMaxValue

---

## Summary

```
#include "cobjects.h"  
#include "julmac.h"
```

```
Bool      JulIsMaxValue( pJul )  
PJUL      pJul;
```

Public Function

A macro is available for this function

## Purpose

The JulIsMaxValue function determines if *JulianTime* pJul is equal to the maximum julian time value.

## Parameter - Description

---

pJul - Pointer to a structure of type *JulianTime*.

## Return Value

The return value from the JulIsMaxValue function is True if pJul is the maximum julian time value, or False if not.

## Notes

Use the JulSetMaxDate function to set the maximum julian time value.

## See Also

JulSetMaxDate

## JullsLeapYear

---

### Example

```
{
Jul jul;
.
.
/* see if a year is a leap year */
if( JulIsLeapYear( 1987 ) == True )
    printf( "Is Leap Year\n" );
else
    printf( "Is Not Leap Year\n" );
}
```

The following output will appear to the terminal:  
Is Not Leap Year

# JullsLeapYear

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Bool      JulIsLeapYear( yr )
MediumInt yr;
```

Public Function

## Purpose

The JulIsLeapYear function determines if the year yr is a leap year.

This function is not typical of a class function in that a *JulianTime* pointer is not the first argument.

## Parameter - Description

---

yr	-	A 4 digit year. [1583 - 4713]
----	---	-------------------------------

## Return Value

The return value from the JulIsLeapYear function is True if the year yr is a leap year and False if yr is not.

### Example

```
{
    Jul    jul;
    MediumInt year;
    MediumInt month;
    MediumInt day;
    Char    dateBuf[11];
    .
    .
    .
    JulInit( &jul ); /* initialized to zero */
    .
    .
    /* create a date string */
    JulToDateStr( &jul, dateBuf, DF_US );
    printf( "%s\n", dateBuf );
    .
}
```

The following output will appear to the terminal:  
1-1-1583



# JulInit

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulInit( pJul )
PJUL      pJul;
```

Public Function

A macro is available for this function

## Purpose

The JulInit function initializes the contents of *JulianTime* pJul to the beginning of the Gregorian calendar (rounded up a year), January 1, 1583, or julian day [2299239].

## Parameter - Description

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
------	---	--

## Return Value

No return value

## JulGetSystemJulianDay

---

### Example

```
{
    Jul      jul;
    MediumInt day;
    MediumInt month;
    MediumInt year;
    .
    .
    /* get the system time */
    JulGetSystemJulianDay( &jul );

    /* print the system date */
    JulConvertJulian( &jul, &year, &month, &day );
    printf( "system date %d-%d-%d\n", month, day, year );
}

The following output will appear to the terminal:
system date [computers date]
```

# JulGetSystemJulianDay

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulGetSystemJulianDay( pJul )
PJUL      pJul;
```

Public Function

## Purpose

The JulGetSystemJulianDay function places the julian time value of the computers clock into *JulianTime* pJul.

## Parameter - Description

---

pJul - Pointer to a structure of type *JulianTime*.

## Return Value

No return value

## Notes

This function uses the standard C function ctime() and converts this returned string into a julian time value.

## Example

```
{
Jul  jul1;
Jul  jul2;
.
.
/* create two julian date */
JulDateStrToJulian( &jul1, "12-30-1987", DF_US );
JulDateStrToJulian( &jul2, "12-23-1987", DF_US );

/* get the number of days between the two dates */
/* jul1 - jul2 = difference */
printf( "Difference = %d\n", JulDiff( &jul1, &jul2 ) );
}
The following output will appear to the terminal
Difference = 7
```

# JulDiffL

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
LargeInt      JulDiffL( pJul1, pJul2 )
PJUL          pJul1;
PJUL          pJul2;
```

Public Function

A macro is available for this function

## Purpose

The JulDiffL function returns *JulianTime* pJul1 minus *JulianTime* pJul2.

## Parameter - Description

---

pJul1	-	Pointer to a structure of type <i>JulianTime</i> .
pJul2	-	Pointer to a structure of type <i>JulianTime</i> .

## Return Value

The return value from the JulDiffL function is the number of days between the two julian time values.

## Notes

This function is used to determine the number of days between two dates. This function should be used if a difference in days in excess of 89 years (days > 32000) is expected.

## See Also

JulDiff

## Example

```
{
Jul  jul1;
Jul  jul2;
.
.

/* create two julian date */
JulDateStrToJulian( &jul1, "12-30-1987", DF_US );
JulDateStrToJulian( &jul2, "12-23-1987", DF_US );

/* get the number of days between the two dates */
/* jul1 - jul2 = difference */
printf( "Difference = %d\n", JulDiff( &jul1, &jul2 ) );
}

The following output will appear to the terminal
Difference = 7
```

# JulDiff

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
MediumInt    JulDiff( pJul1, pJul2 )
PJUL         pJul1;
PJUL         pJul2;
```

Public Function

A macro is available for this function

## Purpose

The JulDiff function returns *JulianTime* pJul1 minus *JulianTime* pJul2.

## Parameter - Description

---

pJul1	-	Pointer to a structure of type <i>JulianTime</i> .
pJul2	-	Pointer to a structure of type <i>JulianTime</i> .

## Return Value

The return value from the JulDiff function is the number of days between two julian time values.

## Notes

This function is used to determine the number of days between two dates. If the difference is in excess of 89 years (days <= SHRT\_MAX or days > SHRT\_MAX) the JulDiffL function should be used.

## See Also

JulDiffL

## JulDaysInYear

---

### Example

```
{
Jul jul;
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* print the days in the above year */
printf( "days in year = %s\n", JulDaysInYear( &jul ) );
}
```

The following output will appear to the terminal:  
days in year = 365



# JulDaysInYear

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
MediumInt    JulDaysInYear( pJul )
PJUL         pJul;
```

Public Function

## Purpose

The JulDaysInYear function returns the number of days in the year of *JulianTime* pJul.

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
------	---	--

## Return Value

The return value from the JulDaysInYear function is the number of days in the year of *JulianTime* pJul.

## See Also

JulDaysInMonth, JulDaysInQuarter

## JulDaysInQuarter

---

### Example

```
{
Jul jul;

/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* print the days in the above quarter */
printf( "days in quarter = %s\n", JulDaysInQuarter( &jul ) );
}
```

The following output will appear to the terminal:  
days in quarter = 92

# JulDaysInQuarter

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
MediumInt    JulDaysInQuarter( pJul )
PJUL         pJul;
```

Public Function

## Purpose

The `JulDaysInQuarter` function returns the number of days in the quarter of *JulianTime* `pJul`.

## Parameter - Description

---

<code>pJul</code>	-	Pointer to a structure of type <i>JulianTime</i> .
-------------------	---	--

## Return Value

The return value from the `JulDaysInQuarter` function is the number of days in the quarter of *JulianTime* `pJul`.

## Notes

Calendar quarters are measured as a year divided into 4 periods of 3 months each (1st period Jan-Mar, 2nd period Apr-Jun, 3rd period Jul-Sep, 4th period Oct-Dec).

## See Also

`JulDaysInMonth`, `JulDaysInYear`

## JulDaysInMonth

---

### Example

```
{
Jul jul;
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* print the days in the above month */
printf( "days in month = %s\n", JulDaysInMonth( &jul ) );
}
```

The following output will appear to the terminal:  
days in month = 31

# JulDaysInMonth

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
MediumInt    JulDaysInMonth( pJul )
PJUL         pJul;
```

Public Function

## Purpose

The `JulDaysInMonth` function returns the number of days in the month of *JulianTime* `pJul`.

## Parameter - Description

---

`pJul` - Pointer to a structure of type *JulianTime*.

## Return Value

The return value from the `JulDaysInMonth` function is the number of days in the month represented by *JulianTime* `pJul`.

## Notes

The return value from the `JulDaysInMonth` function is constant except for February which has 28 or 29 days dependent upon whether it is a leap year or not.

## See Also

`JulDaysInQuarter`, `JulDaysInYear`

## JulDayOfYear

---

### Example

```
{
    Jul jul;

    /* create a julian date */
    JulDateStrToJulian( &jul, "12-23-1987", DF_US );

    /* get the day of the year */
    printf( "Day of year = %d\n", JulDayOfYear( &jul ) );
}
```

The following output will appear to the terminal  
Day of year = 357

# JulDayOfYear

---

## Summary

```
#include "cobjects.h"  
#include "julmac.h"
```

```
MediumInt    JulDayOfYear( pJul )  
PJUL         pJul;
```

Public Function

## Purpose

The JulDayOfYear function returns the day of the year for *JulianTime* pJul.

## Parameter - Description

---

pJul - Pointer to a structure of type *JulianTime*.

## Return Value

The return value from the JulDayOfYear function is the day of the year in the range 1-365 (366 if leap year).

## See Also

JulDayOfWeek

## JulDayOfWeek

---

### Example

```
{
Jul jul;
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* get the day of the week */
printf( "Day of week = %d\n", JulDayOfWeek( &jul ) );
}
```

The following output will appear to the terminal  
Day of week = 3



# JulDayOfWeek

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
MediumInt    JulDayOfWeek( pJul )
PJUL         pJul;
```

Public Function

A macro is available for this function

## Purpose

The JulDayOfWeek function returns the day of week number for the *JulianTime* pJul.

---

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
------	---	--

## Return Value

The return value from the JulDayOfWeek function is the day of week number in the range 0-6.

range:

- 0 - Sunday
- 1 - Monday
- 2 - Tuesday
- 3 - Wednesday
- 4 - Thursday
- 5 - Friday
- 6 - Saturday

## See Also

JulDayOfYear, JulToDateStr

## JulDateStrToJulian

---

### See Also

JulToDateStr, JulValidateDate

### Example

```
{
    Jul      jul;
    MediumInt year;
    MediumInt month;
    MediumInt day;
    .
    .
    .
    /* create an JulianTime pointer from a date */
    JulDateStrToJulian( &jul, "12-23-1987", DF_US );

    /* convert the JulianTime pointer back */
    JulToCalendar( &jul, &year, &month, &day );
    printf( "date %d-%d-%d\n", month, day, year );
}
```

The following output will appear to the terminal:  
date 12-23-1987

# JulDateStrToJulian

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulDateStrToJulian( pJul, pStr, format )
PJUL      pJul;
PSTR      pStr;
DateFormat format;
```

Public Function

## Purpose

The `JulDateStrToJulian` function creates *JulianTime* `pJul` from the *String* `pStr` which contains a date. The format of the date in the string is controlled by passing a date format identifier.

## Parameter - Description

---

<code>pJul</code>	-	Pointer to a structure of type <i>JulianTime</i> .
<code>pStr</code>	-	Contains the string of the specified format.
<code>format</code>	-	Date string format.

date format:

```
DF_US      > MM-DD-YYYY
DF_EUROPE  > DD-MM-YYYY
DF_MILITARY > YYYY-MM-DD
```

## Return Value

No return value

## Notes

This function does no error checking and it is presumed that the caller uses `JulValidateDate` for error checking.

This routine is used to directly convert date strings to julian time.

## Example

```
{
Jul julS;
Jul julD;
Char dateBuf[11];
.
.

/* create a julian date */
JulDateStrToJulian( &julS, "12-23-1987", DF_US );

/* copy the value */
JulCopy( &julD, &julS );

/* lets print the copy - its the same! */
JulToDateStr( &julD, dateBuf, DF_US );
printf( "Copied date %s\n", dateBuf );
}
```

The following output will appear to the terminal  
Copied date 12-23-1987

# JulCopy

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulCopy( pJulD, pJulS )
PJUL      pJulD;
PJUL      pJulS;
```

Public Function

A macro is available for this function

## Purpose

The JulCopy function copies the contents of the source *JulianTime* pJulS to the destination *JulianTime* pJulD.

Parameter	-	Description
-----------	---	-------------

---

pJulD	-	Destination pointer to a structure of type <i>JulianTime</i> .
pJulS	-	Source pointer to a structure of type <i>JulianTime</i> .

## Return Value

No return value

### See Also

JulToCalendar

### Example

```
{
    Jul    jul;
    Char   dateBuf[11];
    :
    :

    /* create the JulianTime pointer */
    JulCalendarToJulian( &jul, 1987, 1, 31 );

    /* create a date string */
    JulToDateStr( &jul, dateBuf, DF_US );
    printf( "%s\n", dateBuf );
}

The following output will appear to the terminal:
1-31-1987
```

# JulCalendarToJulian

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulCalendarToJulian( pJul, year, month, day )
PJUL          pJul;
MediumInt     year;
MediumInt     month;
MediumInt     day;
```

Public Function

## Purpose

The JulCalendarToJulian function converts the values of year, month, and day to the *JulianTime* pJul.

## Parameter - Description

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
year	-	A 4 digit year. [1583 - 4713]
month	-	A 1-2 digit month. [1 - 12]
day	-	A 1-2 digit day. [1 - 31]

## Return Value

No return value

## Notes

This function does no error checking and it is presumed that the caller uses JulValidateDate for error checking.

This function is used to create a julian time value which can be later used by other *JulianTime* routines.

[Year must be in the range [1583:4713].]

[Month must be in the range [1:12].]

[Day must be a valid day for the month and year specified.]

## JulAddYears

---

### Example

```
{
Jul jul;
Char dateBuf[11];
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* subtract 2 years from this date */
JulAddYears( &jul, (-2) );

/* convert the julian date to a string */
JulToDateStr( &jul, dateBuf, DF_US );
printf( "This is the date %s\n", dateBuf );
}
```

The following output will appear to the terminal:  
This is the date 12-23-1985



# JulAddYears

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulAddYears( pJul, yr )
PJUL          pJul;
MediumInt     yr;
```

Public Function

## Purpose

The JulAddYears function adds to or subtracts from *JulianTime* pJul a number of years yr.

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
yr	-	The number of years to be added. If negative then the number of years are subtracted.

## Return Value

No return value

## See Also

JulAddDays, JulAddDaysL, JulAddMonths, JulAddQuarters

## JulAddQuarters

---

### Example

```
{
Jul jul;
Char dateBuf[11];
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* subtract 2 quarters from this date */
JulAddQuarters( &jul, (-2) );

/* convert the julian date to a string */
JulToDateStr( &jul, dateBuf, DF_US );
printf( "This is the date %s\n", dateBuf );
.
}
The following output will appear to the terminal:
This is the date 6-23-1987
```

# JulAddQuarters

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulAddQuarters( pJul, qtrs )
PJUL          pJul;
MediumInt     qtrs;
```

Public Function

## Purpose

The JulAddQuarters function adds to or subtracts from *JulianTime* pJul a number of calendar quarters qtrs.

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
qtrs	-	The number of calendar quarters to be added. If negative then the number of calendar quarters are subtracted.

## Return Value

No return value

## Notes

Calendar quarters are measured as a year divided into 4 periods of 3 months each (1st period Jan-Mar, 2nd period Apr-Jun, 3rd period Jul-Sep, 4th period Oct-Dec).

## See Also

JulAddDays, JulAddDaysL, JulAddMonths, JulAddYears

## JulAddMonths

---

### Example

```
{
Jul jul;
Char dateBuf[11];
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* add 1 month to date */
JulAddMonths( &jul, 1 );

/* convert the julian date to a string */
JulToDateStr( &jul, dateBuf, DF_US );
printf( "This is the date %s\n", dateBuf );
}
The following output will appear to the terminal:
This is the date 1-23-1988
```

# JulAddMonths

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulAddMonths( pJul, mnth )
PJUL          pJul;
MediumInt     mnth;
```

Public Function

## Purpose

The JulAddDays function adds to or subtracts from *JulianTime* pJul a number of months mnth.

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
mnth	-	The number of months to be added. If negative then the number of months are subtracted.

## Return Value

No return value

## Notes

If a date overruns the end of the month the last day of the month is used.

## See Also

JulAddDays, JulAddDaysL, JulAddQuarters, JulAddYears

## JulAddDaysL

---

### Example

```
{
Jul jul;
Char dateBuf[11];
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* subtract 250 years from date */
JulAddDaysL( &jul, (-91250) );

/* convert the julian date to a string */
JulToDateStr( &jul, dateBuf, DF_US );
printf( "This is the date %s\n", dateBuf );
}
The following output will appear to the terminal:
This is the date 2-21-1738
```

# JulAddDaysL

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void          JulAddDaysL( pJul, dy )
PJUL          pJul;
LargeInt      dy;
```

Public Function

A macro is available for this function

## Purpose

The JulAddDaysL function adds to or subtracts from *JulianTime* pJul a number of days dy.

---

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
dy	-	The number of days to be added. If negative then the number of days are subtracted.

## Return Value

No return value

## Notes

Use this function if numbers exceeding 89 years (dy > 32,767 days) are encountered.

## See Also

JulAddDays, JulAddMonths, JulAddQuarters, JulAddYears

## JulAddDays

---

### Example

```
{
Jul jul;
Char dateBuf[11];
.
.
/* create a julian date */
JulDateStrToJulian( &jul, "12-23-1987", DF_US );

/* subtract 10 days from date */
JulAddDays( &jul, (-10) );

/* convert the julian date to a string */
JulToDateStr( &jul, dateBuf, DF_US );
printf( "This is the date %s\n", dateBuf );
}
The following output will appear to the terminal:
This is the date 12-13-1987
```



# JulAddDays

---

## Summary

```
#include "cobjects.h"
#include "julmac.h"
```

```
Void      JulAddDays( pJul, dy )
PJUL      pJul;
MediumInt dy;
```

Public Function

A macro is available for this function

## Purpose

The JulAddDays function adds to or subtracts from *JulianTime* pJul a number of days dy.

Parameter	-	Description
-----------	---	-------------

---

pJul	-	Pointer to a structure of type <i>JulianTime</i> .
dy	-	The number of days to be added. If negative then the number of days are subtracted.

## Return Value

No return value

## Notes

If values greater than 89 years (dy > SHRT\_MAX) are expected the writer should use the JulAddDaysL function.

## See Also

JulAddDaysL, JulAddMonths, JulAddQuarters, JulAddYears

## Class Reference for *JulianTime*

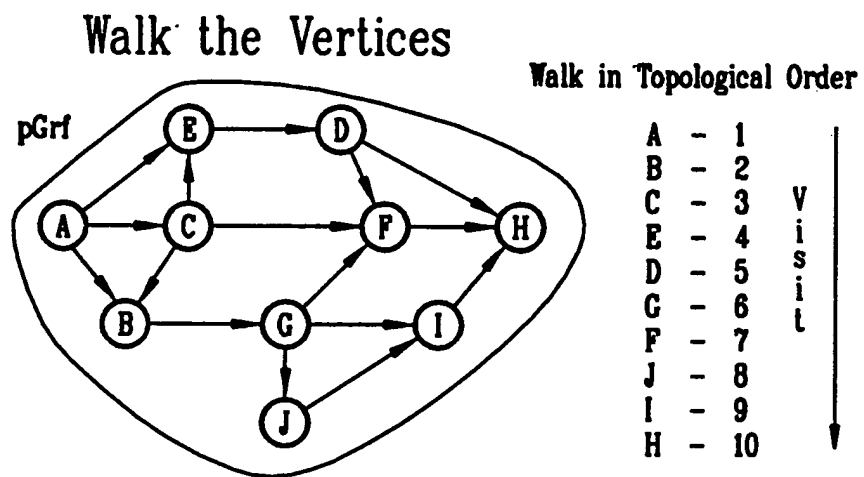
**Structure Name:** JulianTime  
**Abbreviation:** Jul  
**Class Type:** Primitive

# GrfVisitVtxClientInTopOrderFwd

## See Also

GrfEdgClient, GrfVisitVtxClient, GrfVisitVtxClientInTopOrderBwd

## Diagram



# GrfVisitVtxClientInTopOrderFwd

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Void          GrfVisitVtxClientInTopOrderFwd( pGrf, offset, pBlk )
PGRF          pGrf;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The `GrfVisitVtxClientInTopOrderFwd` function walks all the vertices of the *Graph* `pGrf` and calls a *Vertex* client function for each vertex visited. The graph is walked in forward topological order.

The *Block* `pBlk` contains the client function and an optional list of arguments.

## Parameter - Description

---

<code>pGrf</code>	-	Pointer to a structure of type <i>Graph</i> .
<code>offset</code>	-	The distance in bytes between a <i>Vertex</i> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

To use the `GrfVisitVtxClientInTopOrderFwd` function the graph must first be sorted using the `GrfDoTopologicalSort` function.

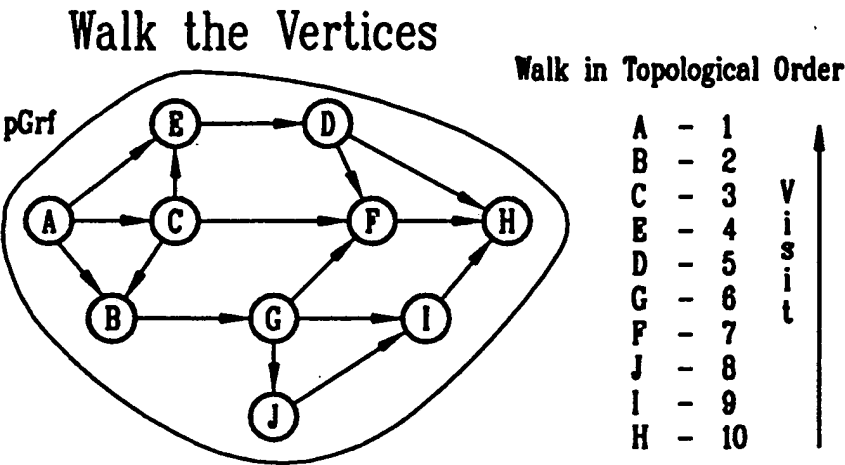
The client function may return a value but it is ignored.

# GrfVisitVtxClientInTopOrderBwd

## See Also

GrfVisitEdgClient, GrfVisitVtxClient, GrfVisitVtxClientInTopOrderFwd

## Diagram



# GrfVisitVtxClientInTopOrderBwd

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Void          GrfVisitVtxClientInTopOrderBwd( pGrf, offset, pBlk )
PGRF          pGrf;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The GrfVisitVtxClientInTopOrderBwd function walks all the vertices of the *Graph* pGrf and calls a *Vertex* client function each vertex visited. The graph is walked in backward topological order.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
offset	-	The distance in bytes between a <i>Vertex</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

To use the GrfVisitVtxClientInTopOrderBwd function the graph must first be sorted using the GrfDoTopologicalSort function.

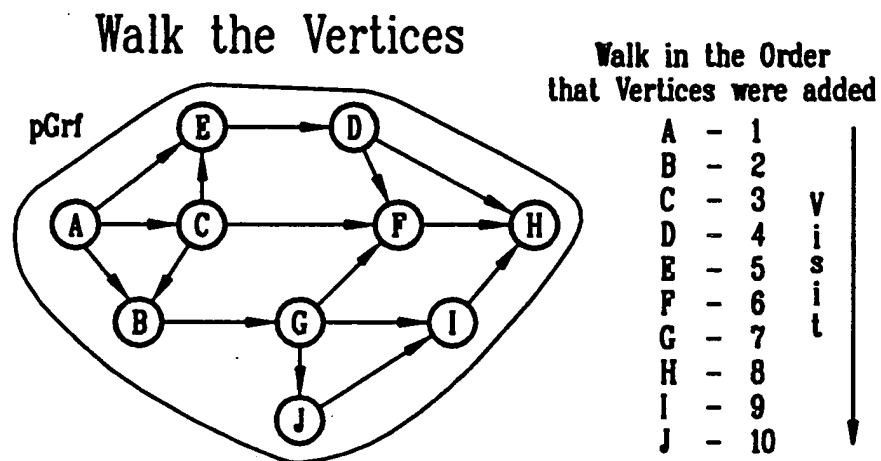
The client function may return a value but it is ignored.

# GrfVisitVtxClient

## See Also

GrfVisitEdgClient, GrfVisitVtxClientInTopOrderBwd,  
GrfVisitVtxClientInTopOrderFwd

## Diagram



## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Void          GrfVisitVtxClient( pGrf, offset, pBlk )
PGRF          pGrf;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The GrfVisitVtxClient function walks the clients of the vertices of the *Graph* pGrf and calls a *Vertex* client function for each vertex visited. The *Block* pBlk contains the client function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
offset	-	The distance in bytes between a <i>Vertex</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The walk of the vertices is in the order that they were added.

The client function may return a value but it is ignored.

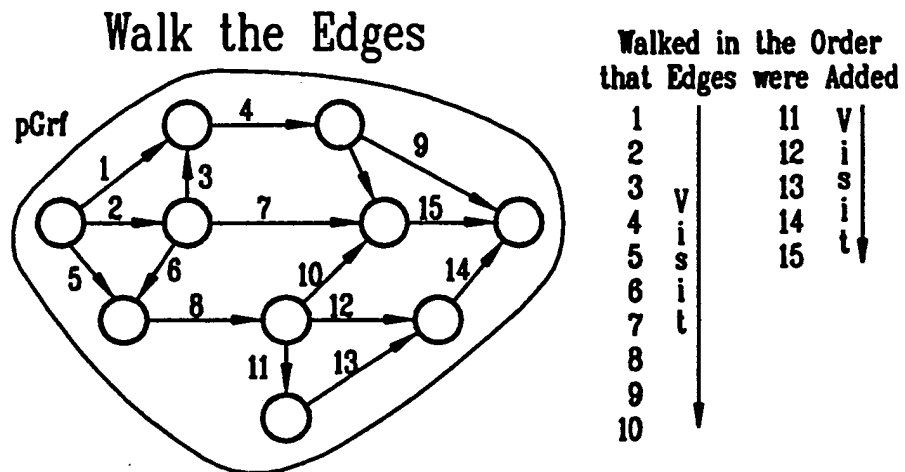


# GrfVisitEdgClient

## See Also

GrfVisitVtxClient, GrfVisitVtxClientInTopOrderBwd,  
GrfVisitVtxClientInTopOrderFwd

## Diagram



# GrfVisitEdgClient

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Void      GrfVisitEdgClient( pGrf, offset, pBlk )
PGRF      pGrf;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The GrfVisitEdgClient function walks the clients of the edges of the *Graph* pGrf and calls an *Edge* client function for each edge visited. The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
offset	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The walk of the edges is in the order that they were added.

The client function may return a value but it is ignored.

# GrfSendDestroy

---

## Summary

```
#include "cobjects.h"  
#include "grfmac.h"
```

```
Void      GrfSendDestroy( pGrf )  
PGRF      pGrf;
```

Public Function

## Purpose

The GrfSendDestroy function sends a message to the client of the *Graph* pGrf asking it to destroy the graph. The *Graph* client function will receive this message and should deinitialize or destroy the graph. This message function should be included in the *Graph* client message array.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

No return value

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfSendDestroy function.

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Void      GrfInit( pGrf )
PGRF      pGrf;
```

Public Function

## Purpose

The GrfInit function initializes the *Graph* pGrf. The GrfInit function should be the first function called when using the *Graph* class.

Parameter	-	Description
-----------	---	-------------

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

No return value

## Notes

The last function to call when using the *Graph* class is GrfDeInit.

## See Also

GrfDeInit, GrfDestroy

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfInit function.

# GrfGetClient

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
POBJ      GrfGetClient( pGrf, offset )
PGRF      pGrf;
MediumInt offset;
```

Public Function

## Purpose

The GrfGetClient function returns a pointer to the client of the *Graph* pGrf.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
offset	-	The distance in bytes between the <i>Graph</i> pGrf and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the GrfGetClient function is a pointer to the client of the *Graph* pGrf.

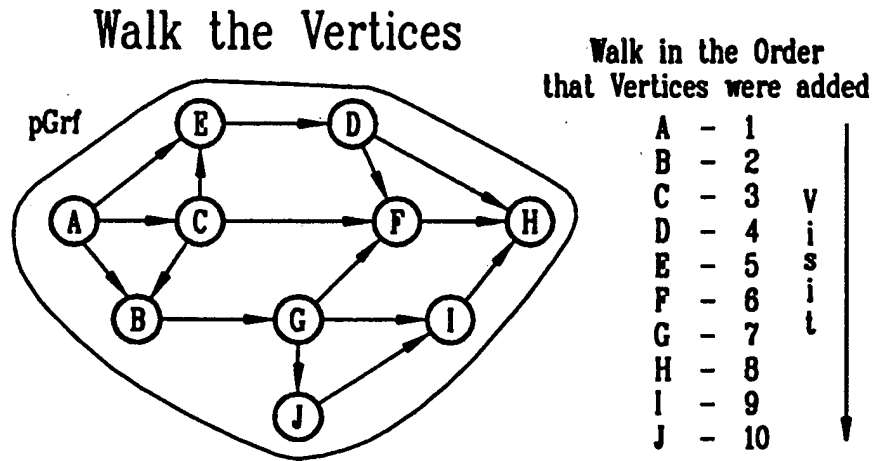
## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfGetClient function.

See Also

GrfFindEdgClient

Diagram



# GrfFindVtxClient

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
POBJ      GrfFindVtxClient( pGrf, offset, pBlk )
PGRF      pGrf;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The GrfFindVtxClient function walks all vertices in the *Graph* pGrf and calls a *Vertex* client function for each vertex traversed. The function terminates when the client function returns True or the last vertex is reached. If a True value is returned, the client pointer of the vertex is returned, otherwise NULL is returned.

The *Block* pBlk contains the client function and an optional list of arguments. The client function must return a boolean (True/False) value.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
offset	-	The distance in bytes between a <i>Vertex</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the GrfFindVtxClient function is a pointer to the *Vertex* client structure for the vertex for which the client function returns True. If no vertex is found then NULL is returned.

## Notes

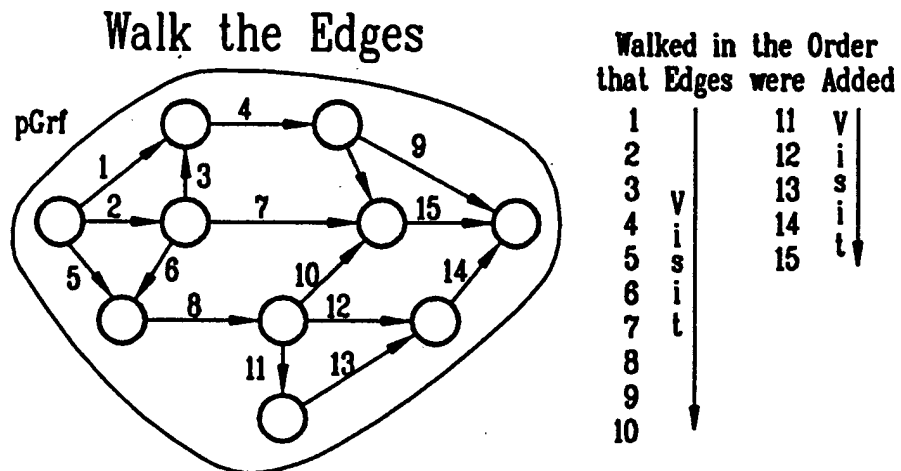
The function should return True for the GrfFindVtxClient function to return True.

## GrfFindEdgClient

See Also

GrfFindVtxClient

Diagram





# GrfFindEdgClient

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
POBJ      GrfFindEdgClient( pGrf, offset, pBlk )
PGRF      pGrf;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The GrfFindEdgClient function walks all edges in the *Graph* pGrf and calls an *Edge* client function for each edge traversed. The function terminates when the client function returns True or the last edge is reached. If a True value is returned, the client pointer of the edge is returned, otherwise NULL is returned.

The *Block* pBlk contains the client function and an optional list of arguments. The client function must return a boolean (True/False) value.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
offset	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

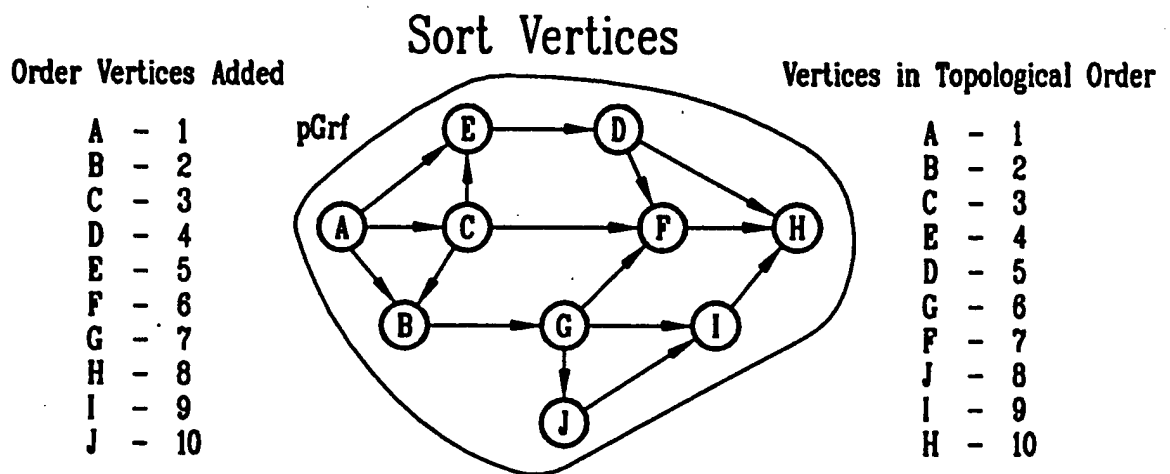
The return value from the GrfFindEdgClient function is a pointer to the *Edge* client structure for the edge for which the client function returns True. If no edge is found then NULL is returned.

## Notes

The function should return True for the GrfFindEdgClient function to return True.

# GrfDoTopologicalSort

## Diagram



# GrfDoTopologicalSort

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Bool          GrfDoTopologicalSort( pGrf )
PGRF          pGrf;
```

Public Function

A macro is available for this function

## Purpose

The GrfDoTopologicalSort function sorts the vertices in the *Graph* pGrf in forward and backward topological order.

## Parameter - Description

---

pGrf - Pointer to a structure of type *Graph*.

## Return Value

The return value from the GrfDoTopologicalSort function is False if the *Graph* pGrf is acyclic or True if it is not.

## Notes

Use the GrfDoTopologicalSort function prior to calling the functions GrfVisitVtxClientInTopOrderBwd or GrfVisitVtxClientInTopOrderFwd. There can be more than one ordering of the vertices which is in topological order.

## See Also

GrfDoBasicTopologicalSort, GrfVisitVtxClientInTopOrderBwd,  
GrfVisitVtxClientInTopOrderFwd

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Void      GrfDestroy( pGrf )
PGRF      pGrf;
```

Public Function

## Purpose

The GrfDestroy function deallocates the memory used by the object and deinitializes the *Graph* object. The *Graph* pGrf should not be referenced after this function call since its memory will have been deallocated.

Any vertices or edges linked to the graph will be unlinked before destroying the object.

Parameter	-	Description
-----------	---	-------------

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

No return value

## Notes

[pGrf must not have a sub-object.]

## See Also

GrfDeInit, GrfInit

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfDestroy function.

# GrfDeInit

---

## Summary

```
#include "cobjects.h"  
#include "grfmac.h"
```

```
Void      GrfDeInit( pGrf )  
PGRF      pGrf;
```

Public Function

## Purpose

The GrfDeInit function deinitializes the *Graph* object. The GrfDeInit function should be the last function called when using the *Graph* class.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

No return value

## Notes

The first function to call when using the *Graph* class is GrfInit.

The vertices and edges can be removed from the graph by calling the GrfClear function.

[pGrf must not contain any vertices.]

[pGrf must not contain any edges.]

## See Also

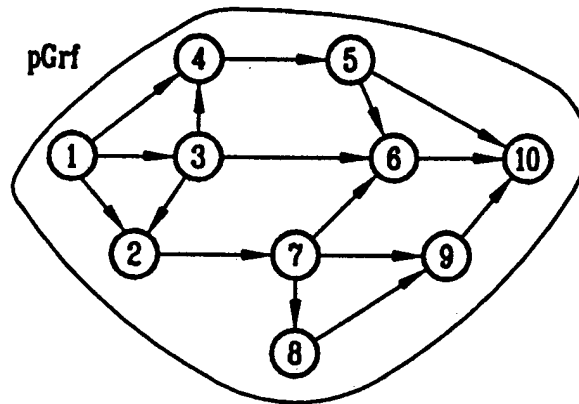
GrfClear, GrfDestroy, GrfInit

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfDeInit function.

Diagram

Number of Vertices = 10



## GrfCountVtx

---

"objects.h"  
"fmac.h"

GrfCountVtx( pGrf )  
pGrf;

in  
available for this function

GrfCountVtx function returns the number of vertices in the *Graph* pGrf.

### - Description

---

- Pointer to a structure of type *Graph*.

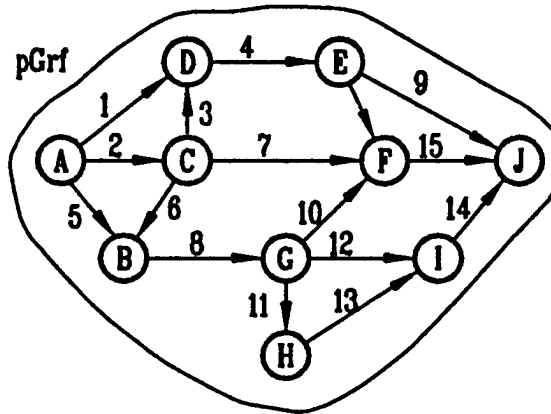
return

value from the GrfCountVtx function is the number of vertices in  
Grf.

g

## Diagram

Number of Edges = 15





# GrfCountEdg

---

## Summary

```
#include "cobjects.h"  
#include "grfmac.h"
```

```
MediumInt    GrfCountEdg( pGrf )  
PGRF         pGrf;
```

Public Function

A macro is available for this function

## Purpose

The GrfCountEdg function returns the number of edges in the *Graph* pGrf.

## Parameter - Description

---

pGrf - Pointer to a structure of type *Graph*.

## Return Value

The return value from the GrfCountEdg function is the number of edges in the *Graph* pGrf.

## See Also

GrfCountVtx

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Void      GrfClear( pGrf )
PGRF      pGrf;
```

Public Function

## Purpose

The GrfClear function unlinks all vertices and edges from the *Graph* pGrf. The topological sort arrays are also cleared.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

No return value

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfClear function.

# GrfAsVtxDll

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
PDLL      GrfAsVtxDll( pGrf )
PGRF      pGrf;
```

Friend Function

## Purpose

The return value from the GrfAsVtxDll function is a pointer to the *List* structure contained by the *Graph* pGrf. The list contains the vertices of pGrf.

## Parameter - Description

---

pGrf - Pointer to a structure of type *Graph*.

## Return Value

The return value from the GrfAsEdgDll function is a pointer to the structure of type *List* contained by the *Graph* class. The list contains the vertices of pGrf.

## See Also

GrfAsEdgDll, GrfAsObj

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfAsVtxDll function.

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
POBJ      GrfAsObj( pGrf )
PGRF      pGrf;
```

Private Function

## Purpose

The GrfAsObj function returns a pointer to the *Object* structure contained by the *Graph* pGrf.

Parameter	-	Description
-----------	---	-------------

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

The return value from the GrfAsObj function is a pointer to the *Object* structure contained by the *Graph* class.

## Notes

The *Object* pointer can be used to send a message to the client of the *Graph*.

## See Also

GrfAsEdgDll, GrfAsVtxDll

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfAsObj function.

# GrfAsEdgDll

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
PDLL      GrfAsEdgDll( pGrf )
PGRF      pGrf;
```

Friend Function

## Purpose

The return value from the GrfAsEdgDll function is a pointer to the *List* structure contained by the *Graph* pGrf. The list contains the edges of pGrf.

## Parameter - Description

---

pGrf - Pointer to a structure of type *Graph*.

## Return Value

The return value from the GrfAsEdgDll function is a pointer to the structure of type *List* contained by the *Graph* class. The list contains the edges of pGrf.

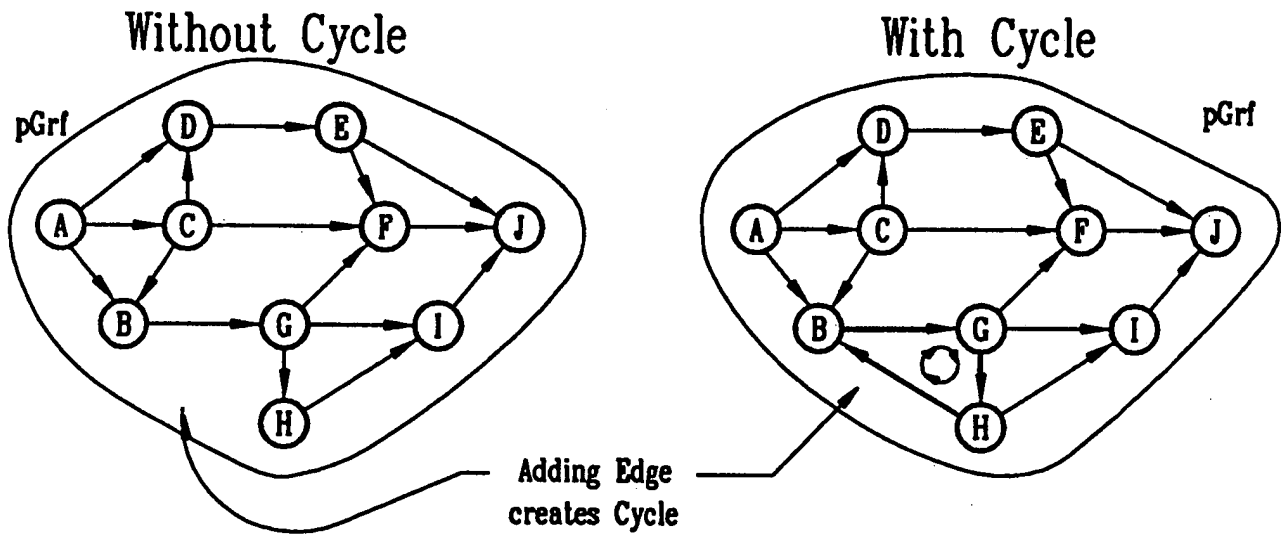
## See Also

GrfAsObj, GrfAsVtxDll

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the GrfAsEdgDll function.

## Diagram



# GrfAnyCycles

---

## Summary

```
#include "cobjects.h"
#include "grfmac.h"
```

```
Bool      GrfAnyCycles( pGrf )
PGRF      pGrf;
```

Public Function

A macro is available for this function

## Purpose

The GrfAnyCycles function determines if the graph is acyclic.

## Parameter - Description

---

pGrf	-	Pointer to a structure of type <i>Graph</i> .
------	---	---

## Return Value

The return value from the GrfAnyCycles function is False if the *Graph* pGrf is acyclic or True if it is not.

## Class Reference for *Graph*

**Structure Name:** Graph  
**Abbreviation:** Grf  
**Class Type:** Inheritable class

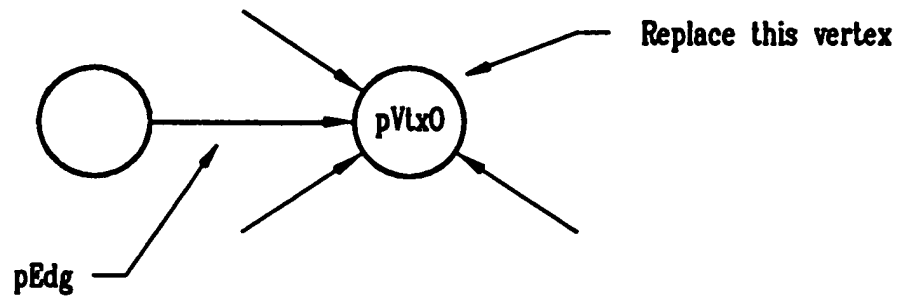


# EdgUpdateOutVtx

---

## Diagram

Update the outgoing Vertex



Legend

# EdgUpdateOutVtx

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgUpdateOutVtx( pEdg, pVtxO )
PEDG      pEdg;
PVTX      pVtxO;
```

Public Function

## Purpose

The EdgUpdateOutVtx function replaces the current outgoing vertex for the *Edge* pEdg with the *Vertex* pVtxO.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
pVtxO	-	Pointer to a structure of type <i>Vertex</i> where this is the outgoing vertex. An outgoing vertex has the edge as an incoming edge.

## Return Value

No return value

## Notes

[pEdg must be linked to a graph.]

[pEdg must be linked to two vertices.]

[The *Vertex* pVtxO must be linked to a graph.]

## See Also

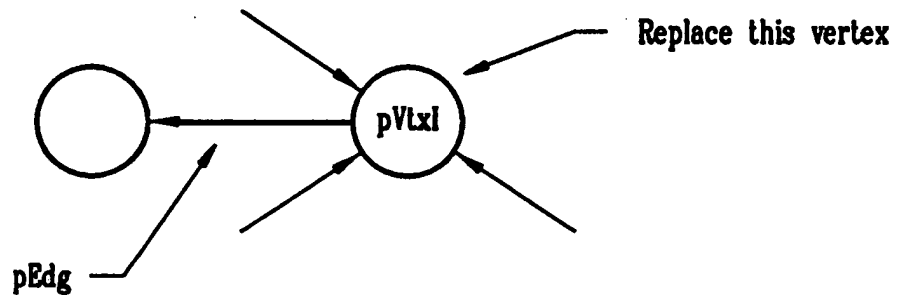
EdgUpdateInVtx

# EdgUpdateInVtx

---

## Diagram

Update the incoming Vertex



Legend

## EdgUpdateInVtx

---

### Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgUpdateInVtx( pEdg, pVtxI )
PEDG      pEdg;
PVTX      pVtxI;
```

Public Function

### Purpose

The EdgUpdateInVtx function replaces the current incoming vertex for the *Edge* pEdg with the *Vertex* pVtxI.

---

Parameter	-	Description
-----------	---	-------------

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
pVtxI	-	Pointer to a structure of type <i>Vertex</i> where this is the incoming vertex. An incoming vertex has the edge as an outgoing edge.

### Return Value

No return value

### Notes

[pEdg must be linked to a graph.]

[pEdg must be linked to two vertices.]

[The *Vertex* pVtxI must be linked to a graph.]

### See Also

EdgUpdateOutVtx

# EdgSendDestroy

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgSendDestroy( pEdg )
PEDG      pEdg;
```

Public Function

A macro is available for this function

## Purpose

The EdgSendDestroy function sends a message to the client of the *Edge* pEdg asking it to destroy the edge. The *Edge* client function will receive this message and should deinitialize or destroy the edge. This message function should be included in the *Edge* client message array.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

No return value

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgSendDestroy function.

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgInit( pEdg )
PEDG      pEdg;
```

Public Function

## Purpose

The EdgInit function initializes the *Edge* object. The EdgInit function should be the first function called when using the *Edge* class.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

No return value

## Notes

The last function to call when using the *Edge* class is EdgDeInit.

## See Also

EdgDeInit, EdgDestroy

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgInit function.

# EdgInGrf

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Bool      EdgInGrf( pEdg )
PEDG      pEdg;
```

Public Function

A macro is available for this function

## Purpose

The EdgInGrf function determines if the *Edge* pEdg is linked to a *Graph*.

## Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

## Return Value

The return value from the EdgInGrf function is True if the *Edge* pEdg is linked to a *Graph*, otherwise False is returned.

## See Also

EdgGetGrf

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgInGrf function.

# EdgHasVertices

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Bool      EdgHasVertices( pEdg )
PEDG      pEdg;
```

Public Function

## Purpose

The EdgHasVertices function determines if the *Edge* pEdg has any vertices linked to it.

Parameter	-	Description
-----------	---	-------------

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

The return value from the EdgHasVertices function is True if the *Edge* pEdg has any vertices, otherwise False is returned.

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgHasVertices function.

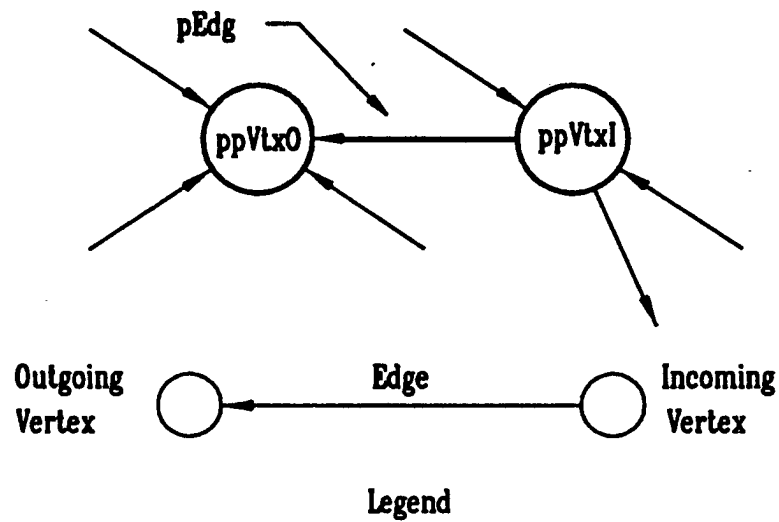


# EdgGetVertices

---

## Diagram

Return both Vertices



# EdgGetVertices

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgGetVertices( pEdg, ppVtxI, ppVtxO )
PEDG      pEdg;
PVTX      *ppVtxI;
PVTX      *ppVtxO;
```

Public Function

A macro is available for this function

## Purpose

The EdgGetVertices function returns the incoming *Vertex* pVtxI and the outgoing *Vertex* pVtxO for the *Edge* pEdg.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
ppVtxI	-	Returned pointer to a structure of type pointer to <i>Vertex</i> where this is the incoming vertex. An incoming vertex has the edge as an outgoing edge.
ppVtxO	-	Returned pointer to a structure of type pointer to <i>Vertex</i> where this is the outgoing vertex. An outgoing vertex has the edge as an incoming edge.

## Return Value

No return value

## See Also

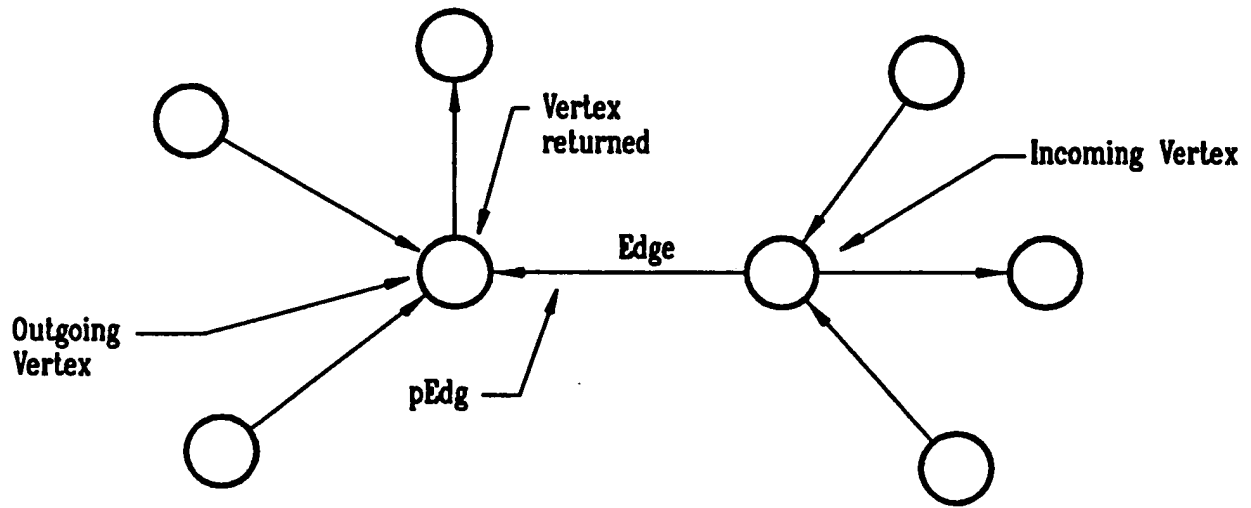
EdgGetInVtx, EdgGetOutVtx

# EdgGetOutVtx

---

## Diagram

Get outgoing Vertex



## EdgGetOutVtx

---

### Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
PVTX      EdgGetOutVtx( pEdg )
PEDG      pEdg;
```

#### Friend Function

A macro is available for this function

### Purpose

The EdgGetOutVtx function returns a pointer to the outgoing *Vertex* that the *Edge* pEdg is linked to.

### Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

### Return Value

The return value from the EdgGetOutVtx function is a pointer to a *Vertex* structure. This vertex is the outgoing *Vertex* to pEdg. An outgoing vertex has the edge as an incoming edge.

### See Also

EdgGetInVtx



# EdgGetNextOut

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
PEDG      EdgGetNextOut( pEdg )
PEDG      pEdg;
```

Friend Function

A macro is available for this function

## Purpose

The EdgGetNextOut function returns a pointer to the successor outgoing *Edge* from the *Edge* pEdg or NULL if none exists. This successor edge is the next edge in a list of outgoing edges for a vertex.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

The return value from the EdgGetNextIn function is a pointer to an *Edge* structure or NULL. This edge is the successor outgoing edge to the *Edge* pEdg.

## See Also

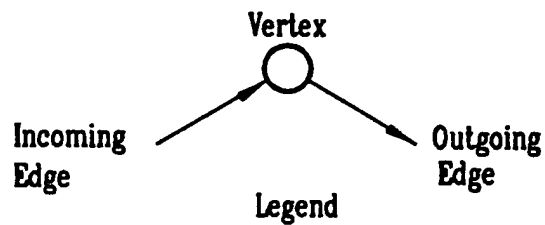
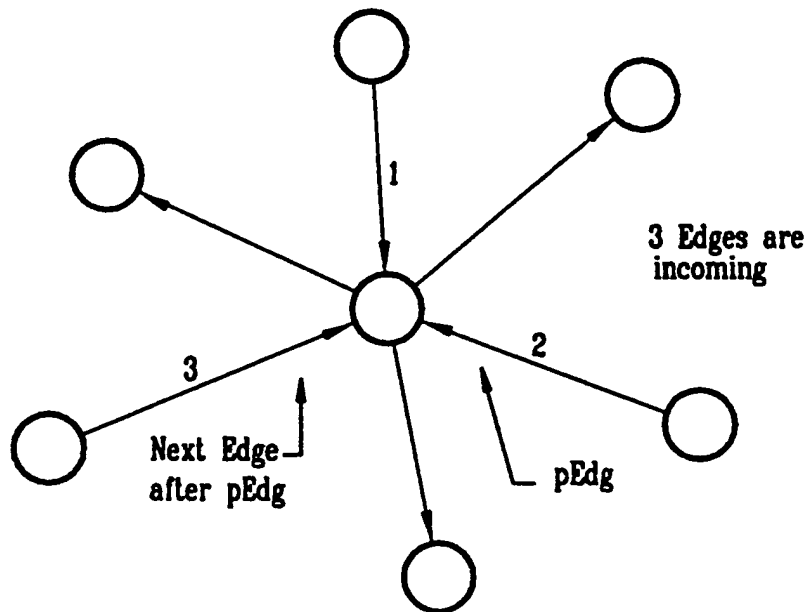
EdgGetNextIn

# EdgGetNextIn

---

## Diagram

Get next incoming Edge for Vertex to pEdg



# EdgGetNextIn

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
PEDG      EdgGetNextIn( pEdg )
PEDG      pEdg;
```

Friend Function

A macro is available for this function

## Purpose

The EdgGetNextIn function returns a pointer to the successor incoming *Edge* from the *Edge* pEdg or NULL if none exists. This successor edge is the next edge in a list of incoming edges for a vertex.

## Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

## Return Value

The return value from the EdgGetNextIn function is a pointer to an *Edge* structure or NULL. This edge is the successor incoming edge to the *Edge* pEdg.

## See Also

EdgGetNextOut

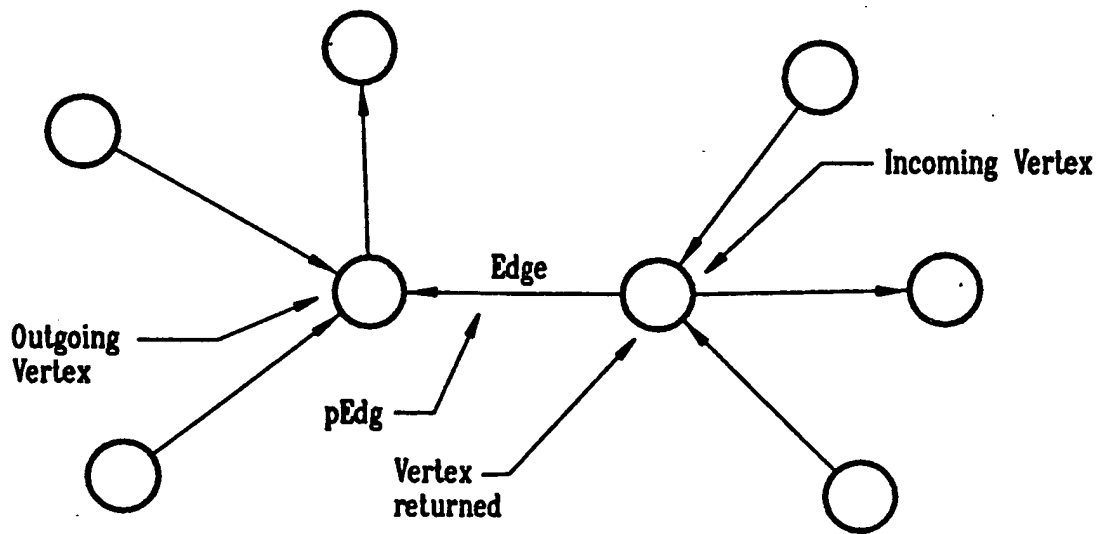


# EdgGetInVtx

---

## Diagram

Get incoming Vertex



## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
PVTX      EdgGetInVtx( pEdg )
PEDG      pEdg;
```

Friend Function

A macro is available for this function

## Purpose

The EdgGetInVtx function returns a pointer to the incoming *Vertex* that the *Edge* pEdg is linked to.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

The return value from the EdgGetInVtx function is a pointer to a *Vertex* structure, otherwise NULL is returned. This vertex is the incoming *Vertex* to pEdg. An incoming vertex has the edge as an outgoing edge.

## See Also

EdgGetOutVtx, EdgGetVertices

# EdgGetGrf

---

## Summary

#include "cobjects.h"

#include "edgmac.h"

PGRF            EdgGetGrf( pEdg )  
PEDG           pEdg;

Friend Function

A macro is available for this function

## Purpose

The EdgGetGrf function returns a pointer to the graph that the *Edge* pEdg is linked to.

## Parameter   -   Description

---

pEdg           -   Pointer to a structure of type *Edge*.

## Return Value

The return value from the EdgGetGrf function is a pointer to a *Graph* structure or NULL if not linked to a graph.

## See Also

EdgInGrf

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgGetGrf function.

# EdgGetClient

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
POBJ      EdgGetClient( pEdg, offset )
PEDG      pEdg;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The EdgGetClient function returns a pointer to the client of the *Edge* pEdg.

Parameter	-	Description
-----------	---	-------------

pEdg	-	Pointer to a structure of type <i>Edge</i> .
offset	-	The distance in bytes between the <i>Edge</i> pEdg and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

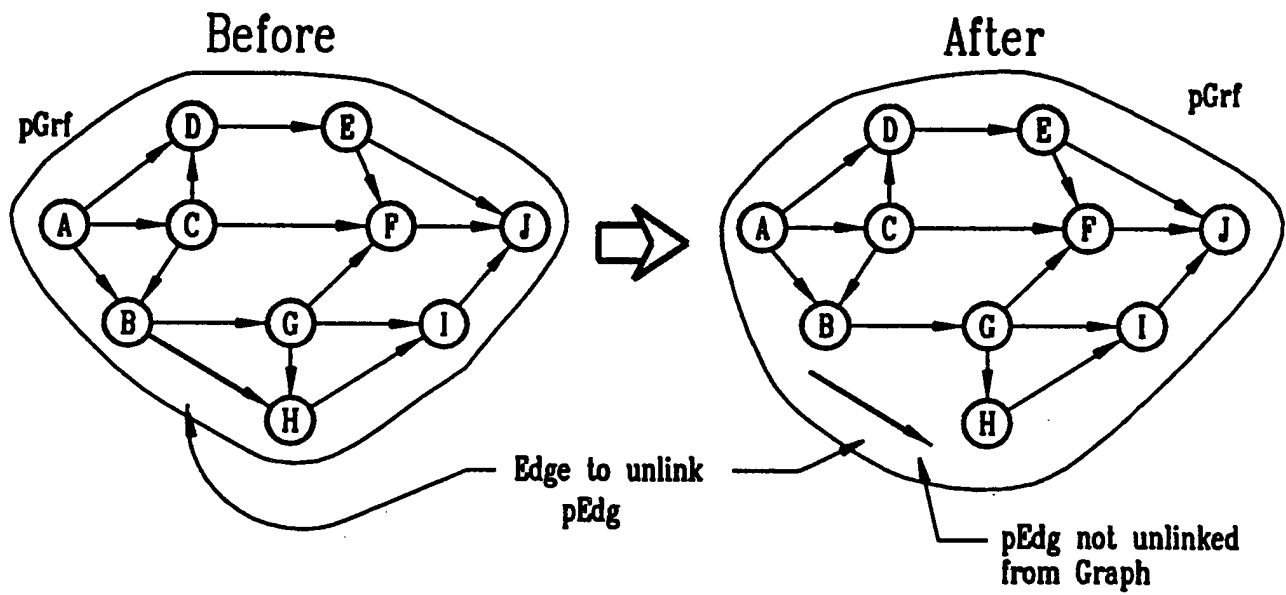
The return value from the EdgGetClient function is a pointer to the client of the *Edge* pEdg.

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgGetClient function.

# EdgDisconnectFromVertices

## Diagram



# EdgDisconnectFromVertices

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgDisconnectFromVertices( pEdg )
PEDG      pEdg;
```

Public Function

## Purpose

The EdgDisconnectFromVertices function unlinks the *Edge* pEdg to its incoming vertex and its outgoing vertex.

## Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

## Return Value

No return value

## Notes

[pEdg must be linked to a graph.]

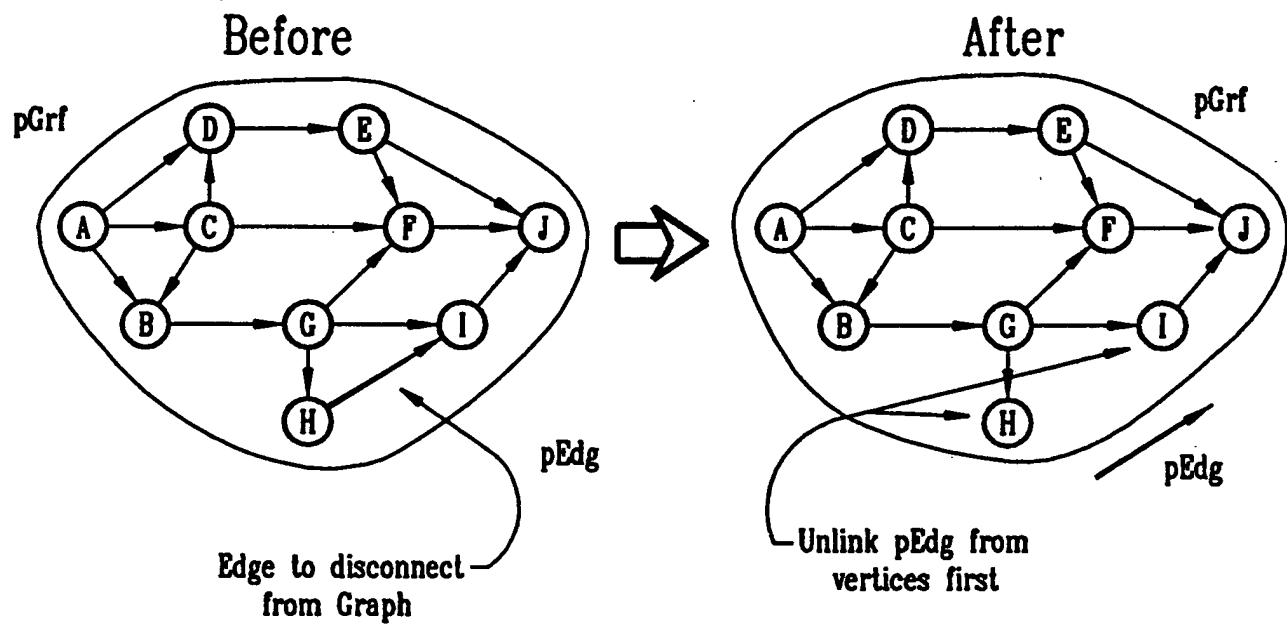
[pEdg must not be linked to any vertices.]

## See Also

EdgConnectToVertices, EdgGetInVtx, EdgGetOutVtx, EdgGetVertices

# EdgDisconnectFromGrf

## Diagram



# EdgDisconnectFromGrf

---

## Summary

#include "cobjects.h"  
#include "edgmac.h"

Void                EdgDisconnectFromGrf( pEdg )  
PEDG                pEdg;

Public Function

A macro is available for this function

## Purpose

The EdgDisconnectFromGrf function unlinks the *Edge* pEdg from it's *Graph*. The function requires that any linkage to vertices be removed prior to the call.

## Parameter   -   Description

---

pEdg               -    Pointer to a structure of type *Edge*.

## Return Value

No return value

## Notes

[pEdg must be linked to a graph.]

[pEdg must not be linked to any vertices.]

## See Also

EdgConnectToGrf, EdgDisconnectFromVertices



# EdgDestroy

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgDestroy( pEdg )
PEDG      pEdg;
```

Public Function

## Purpose

The EdgDestroy function deallocates the memory used by the object and deinitializes the *Edge* object. The *Edge* pEdg should not be referenced after this function call since its memory will have been deallocated.

Any vertices linked to pEdg will be unlinked and pEdg will also be unlinked from its graph.

Parameter	-	Description
-----------	---	-------------

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

No return value

## Notes

[pEdg must not have a sub-object.]

## See Also

EdgDeInit, EdgInit

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgDestroy function.

## Summary

```
#include "objects.h"
#include "edgmac.h"
```

```
Void      EdgDeInit( pEdg )
PEDG      pEdg;
```

Public Function

## Purpose

The EdgDeInit function deinitializes the *Edge* object. The EdgDeInit function should be the last function called when using the *Edge* class.

## Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

## Return Value

No return value

## Notes

The first function to call when using the *Edge* class is EdgInit.

[pEdg must not already be linked to a graph.]

[pEdg must not be linked to any vertices.]

## See Also

EdgDestroy, EdgInit

## Example

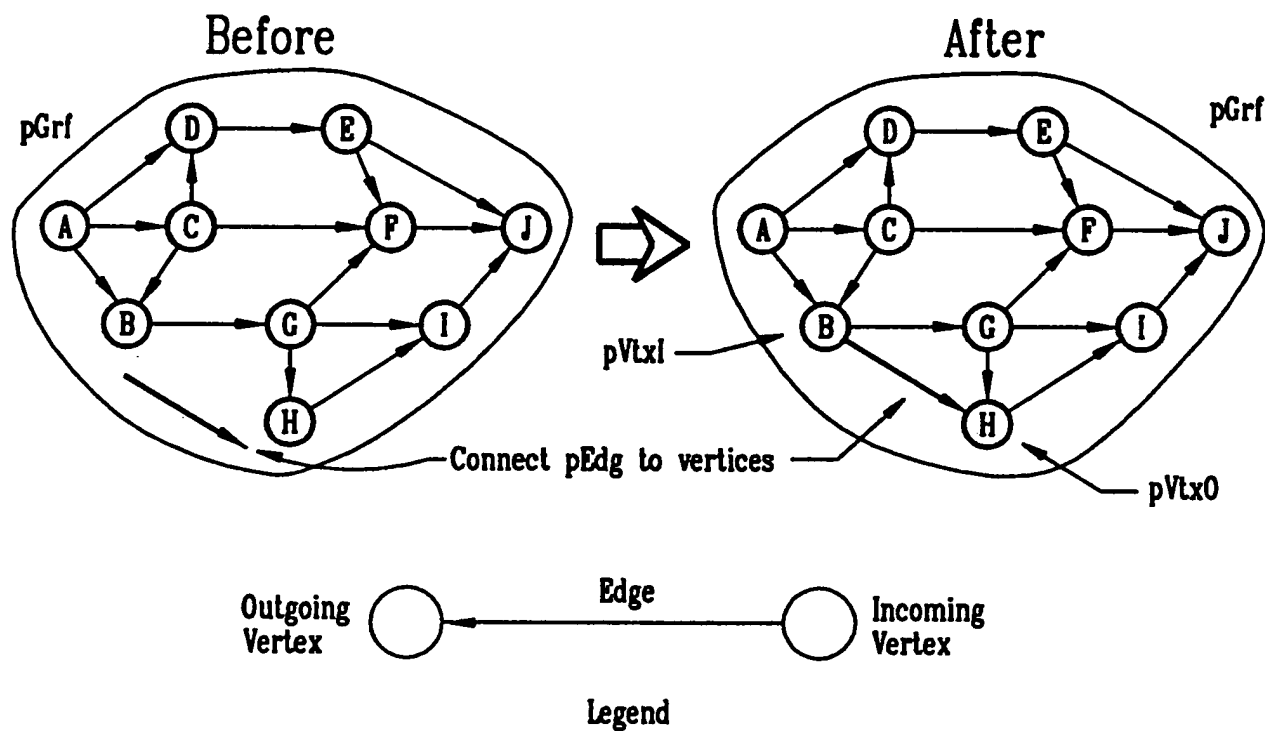
Please refer to class test procedure TSTGRF.C for an example of the use of the EdgDeInit function.

# EdgConnectToVertices

## See Als

EdgDisconnectFromVertices, EdgGetInVtx, EdgGetOutVtx, EdgGetVertices

## Diagram



# EdgConnectToVertices

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgConnectToVertices( pEdg, pVtxI, pVtxO )
PEDG      pEdg;
PVTX      pVtxI;
PVTX      pVtxO;
```

Public Function

## Purpose

The EdgConnectToVertices function links the *Edge* pEdg to the incoming *Vertex* pVtxI and the outgoing *Vertex* pVtxO.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
pVtxI	-	Pointer to a structure of type <i>Vertex</i> where this is the incoming vertex. An incoming vertex has the edge as an outgoing edge.
pVtxO	-	Pointer to a structure of type <i>Vertex</i> where this is the outgoing vertex. An outgoing vertex has the edge as an incoming edge.

## Return Value

No return value.

## Notes

[pEdg must be linked to a graph.]

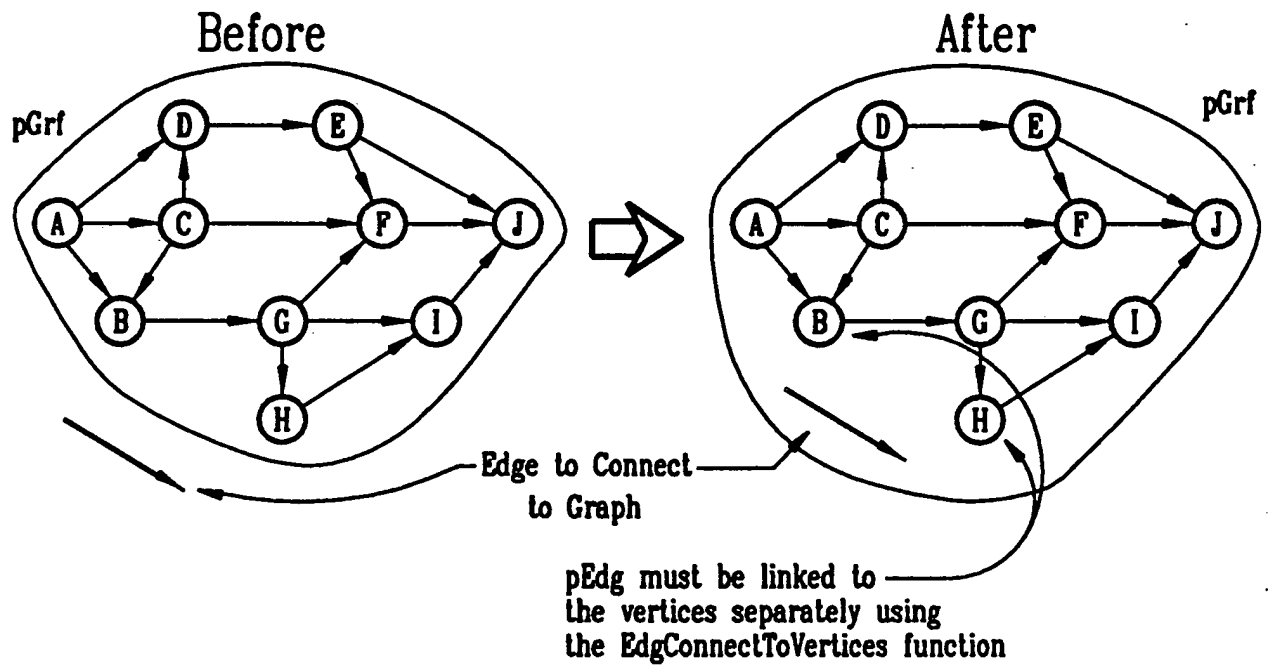
[The *Vertex* pVtxI must be linked to a graph.]

[The *Vertex* pVtxO must be linked to a graph.]

[pEdg must not be linked to any vertices.]

# EdgConnectToGrf

## Diagram



# EdgConnectToGrf

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Void      EdgConnectToGrf( pEdg, pGrf )
PEDG      pEdg;
PGRF      pGrf;
```

### Public Function

A macro is available for this function

## Purpose

The EdgConnectToGrf function links the *Edge* pEdg to the *Graph* pGrf.

Parameter	-	Description
-----------	---	-------------

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
pGrf	-	Pointer to the <i>Graph</i> structure that the edge will be linked to.

## Return Value

No return value

## Notes

[pEdg must not already be linked to a graph.]

[pEdg must not be linked to any vertices.]

## See Also

EdgDisconnectFromGrf

# EdgCompareInVtx

---

## Notes

The client function should return True for the EdgCompareInVtx function to return True.

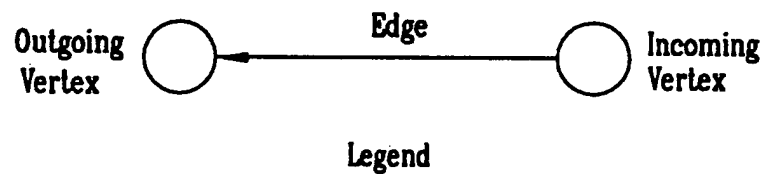
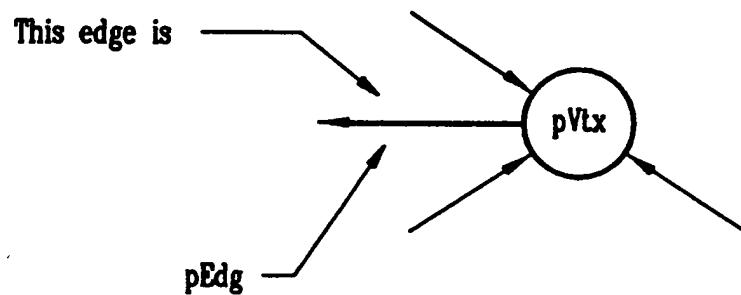
The optional function will only be called if pEdg is linked to pVtx.

## See Also

VtxFindOutEdg, VtxFindOutEdgClient

## Diagram

Is pVtx an incoming Vertex to pEdg ?



# EdgCompareInVtx

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
Bool      EdgCompareInVtx( pEdg, pVtx, offset, pBlk )
PEDG      pEdg;
PVTX      pVtx;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The EdgCompareInVtx function determines if the *Edge* pEdg has an incoming *Vertex* pVtx.

If the caller requires that the comparison should have additional criteria other than linkage, an *Edge* client function can be passed. The function is passed the matching edge to determine if this is the edge in question.

The *Block* pBlk contains the client function and an optional list of arguments. The function must return a boolean (True/False) value.

The EdgCompareInVtx function is typically used as part of a larger function for scanning edges.

Parameter	-	Description
-----------	---	-------------

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
pVtx	-	Pointer to a structure of type <i>Vertex</i> where this is an incoming vertex. An incoming vertex has the edge as an outgoing edge.
offset	-	The distance in bytes between an <i>Edge</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the EdgCompareInVtx function is True if the incoming *Vertex* pVtx is linked to the *Edge* pEdg, otherwise False is returned.



# EdgClear

---

## Summary

```
#include "cobjects.h"  
#include "edgmac.h"
```

```
Void      EdgClear( pEdg )  
PEDG      pEdg;
```

Public Function

## Purpose

The EdgClear function unlinks the *Edge* pEdg from its vertices and also unlinks pEdg from its graph.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

No return value

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgClear function.

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
PLEL      EdgAsOutLel( pEdg )
PEDG      pEdg;
```

Friend Function

A macro is available for this function

## Purpose

The EdgAsInLel function returns a pointer to the outgoing vertex *ListElement* structure contained the *Edge* pEdg.

## Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

## Return Value

The return value from the EdgAsInLel function is a pointer to the outgoing vertex *ListElement* structure contained by the *Edge* pEdg.

## Notes

This function is used by the *Vertex* class.

## See Also

EdgAsGrfLel, EdgAsInLel

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgAsOutLel function.

# EdgAsObj

---

## Summary

```
#include "cobjects.h"  
#include "edgmac.h"
```

```
POBJ      EdgAsObj( pEdg )  
PEDG      pEdg;
```

Private Function

## Purpose

The EdgAsObj function returns a pointer to the *Object* structure contained by the *Edge* pEdg.

## Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

## Return Value

The return value from the EdgAsObj function is a pointer to the *Object* structure contained by the *Edge* class.

## Notes

The *Object* pointer can be used to send a message to the client of the edge.

## See Also

EdgAsGrfLel, EdgAsInLel, EdgAsOutLel

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgAsObj function.

## Summary

```
#include "objects.h"
#include "edgmac.h"
```

```
PLEL      EdgAsInLel( pEdg )
PEDG      pEdg;
```

Friend Function

A macro is available for this function

## Purpose

The EdgAsInLel function returns a pointer to the incoming vertex *ListElement* structure contained the *Edge* pEdg.

## Parameter - Description

---

pEdg - Pointer to a structure of type *Edge*.

## Return Value

The return value from the EdgAsInLel function is a pointer to the incoming vertex *ListElement* structure contained by the *Edge* pEdg.

## Notes

This function is used by the *Vertex* class.

## See Also

EdgAsGrfLel, EdgAsOutLel

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgAsInLel function.

# EdgAsGrfLel

---

## Summary

```
#include "cobjects.h"
#include "edgmac.h"
```

```
PLEL      EdgAsGrfLel( pEdg )
PEDG      pEdg;
```

Friend Function

A macro is available for this function

## Purpose

The EdgAsGrfLel function returns a pointer to the *ListElement* structure that is contained within the *Edge* pEdg. This *ListElement* is used by the *List* within the *Graph* object.

## Parameter - Description

---

pEdg	-	Pointer to a structure of type <i>Edge</i> .
------	---	--

## Return Value

The return value from the EdgAsGrfLel function is a pointer to a *ListElement* structure.

## Notes

The *ListElement* pointer returned is contained as part of the *Edge* pEdg. This function is used by the *Graph* object.

## See Also

EdgAsInLel, EdgAsOutLel

## Example

Please refer to class test procedure TSTGRF.C for an example of the use of the EdgAsGrfLel function.

## Class Reference for *Edge*

**Structure Name:** Edge  
**Abbreviation:** Edg  
**Class Type:** Inheritable class

## ***DynamicArray***

---

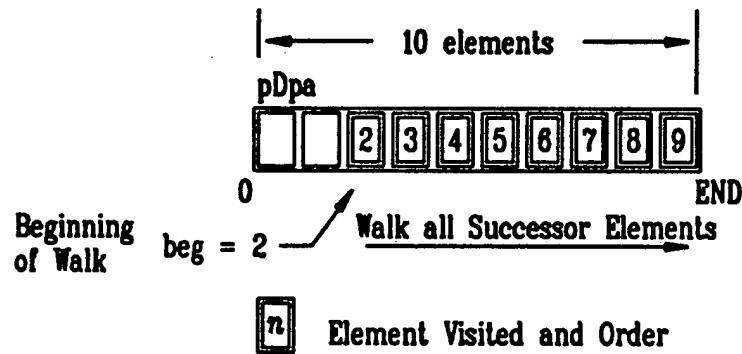
This page is intentionally left blank

# DpaVisitSelfAndSuccessors

## See Also

DpaVisit, DpaVisitClient, DpaVisitRange, DpaVisitRegion

## Diagram





# DpaVisitSelfAndSuccessors

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaVisitSelfAndSuccessors( pDpa, beg, pBlk )
PDPA      pDpa;
MediumInt beg;
PBLK      pBlk;
```

Public Function

## Purpose

The `DpaVisitSelfAndSuccessors` function walks the *DynamicArray* `pDpa` and calls a `GenericPointer` function for each element visited. The array is walked in a forward direction (self to last) starting with the element `beg`.

The *Block* `pBlk` contains the function and an optional list of arguments.

## Parameter - Description

---

<code>pDpa</code>	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
<code>beg</code>	-	Index to start of traverse. [0-END]
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

## Return Value

No return value

## Notes

The `GenericPointer` function may return a value but it is ignored.

[The index `beg` must be in the range [0-END].]

## DpaVisitRegion

### Notes

The GenericPointer function may return a value but it is ignored.

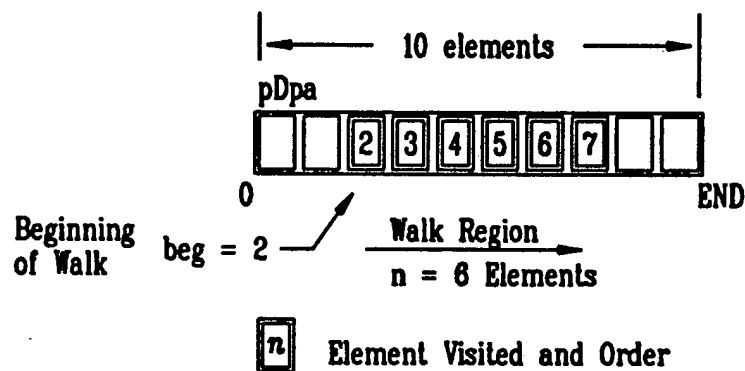
[The index beg must be in the range [0-END].]

[(beg + n) must be less than or equal to the size of the array.]

### See Also

DpaVisit, DpaVisitClient, DpaVisitRange, DpaVisitSelfAndSuccessors

### Diagram



# DpaVisitRegion

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void          DpaVisitRegion( pDpa, beg, n, pBlk )
PDPA          pDpa;
MediumInt     beg;
MediumInt     n;
PBLK          pBlk;
```

Public Function

## Purpose

The DpaVisitRegion function walks a region of array elements, starting with beg for a count of n times, for the *DynamicArray* pDpa and calls a GenericPointer function for each element visited. The array is walked in a forward direction.

The *Block* pBlk contains the function and an optional list of arguments.

A region of array elements is defined as a beginning element and an count from the beginning element. The count cannot be less than 1. See section on region definition for more details.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
beg	-	Index to start of traverse. [0-END]
n	-	Number of elements to traverse. [1-SIZE]
pBlk	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

count: SIZE is the number of elements in the array.

## Return Value

No return value

## DpaVisitRange

### Notes

The GenericPointer function may return a value but it is ignored.

[The index beg must be in the range [0-END].]

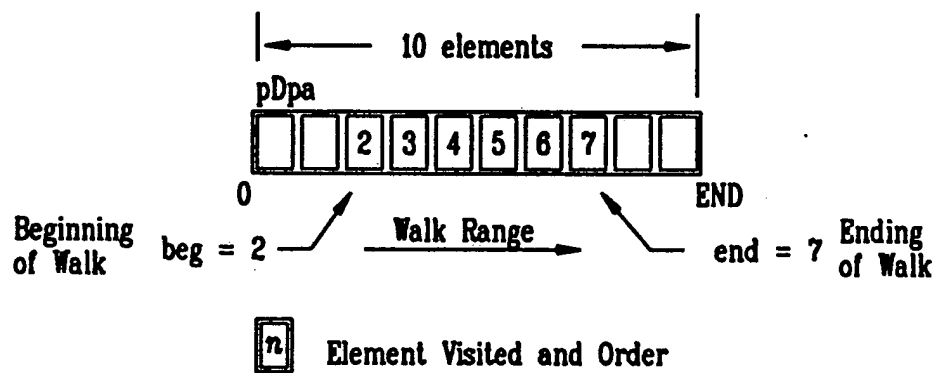
[The index end must be in the range [0-END].]

[beg must be less than or equal to end.]

### See Also

DpaVisit, DpaVisitClient, DpaVisitRegion, DpaVisitSelfAndSuccessors

### Diagram



# DpaVisitRange

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaVisitRange( pDpa, beg, end, pBlk )
PDPA      pDpa;
MediumInt beg;
MediumInt end;
PBLK      pBlk;
```

Public Function

## Purpose

The *DpaVisitRange* function walks a range of array elements, *beg* through *end*, for the *DynamicArray* *pDpa* and calls a *GenericPointer* function for each element visited. The array is walked in a forward direction.

The *Block* *pBlk* contains the function and an optional list of arguments.

A range of array elements is defined as a beginning element and an ending element. The beginning element can equal the ending element. See section on range definition for more details.

## Parameter - Description

---

<i>pDpa</i>	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
<i>beg</i>	-	Index to start of traverse. [0-END]
<i>end</i>	-	Index to end of traverse. [0-END]
<i>pBlk</i>	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

## Return Value

No return value

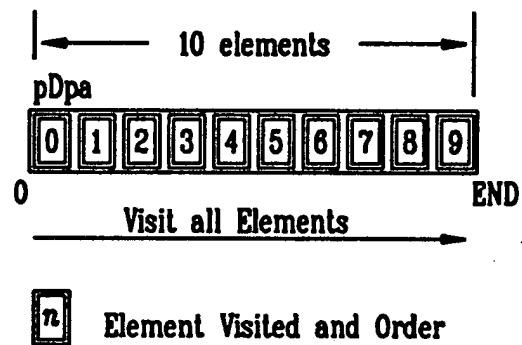
## DpaVisitClient

---

### See Also

DpaVisitRange, DpaVisitRegion, DpaVisitSelfAndSuccessors

### Diagram



# DpaVisitClient

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaVisitClient( pDpa, offset, pBlk )
PDPA      pDpa;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The `DpaVisitClient` function walks the entire *DynamicArray* `pDpa` and calls a client function for each element visited. The array is walked in a forward direction.

The *Block* `pBlk` contains the client function and an optional list of arguments.

## Parameter - Description

---

<code>pDpa</code>	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
<code>offset</code>	-	The distance in bytes between the <i>DynamicArray</i> <code>pDpa</code> and it's client pointer. The value must be 0 or negative.
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

`offset`: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

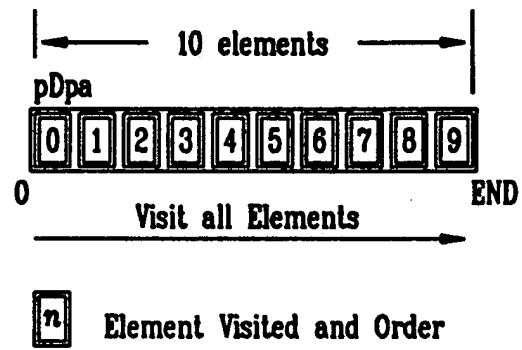
No return value

## Notes

The `GenericPointer` function may return a value but it is ignored.

If any of the elements of the array are `NULL`, they are not visited.

## Diagram





# DpaVisit

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void          DpaVisit( pDpa, pBlk )
PDPA          pDpa;
PBLK          pBlk;
```

Public Function

## Purpose

The DpaVisit function walks the entire *DynamicArray* pDpa and calls a GenericPointer function for each element visited. The array is walked in a forward direction.

The *Block* pBlk contains the GenericPointer function and an optional list of arguments.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

## Return Value

No return value

## Notes

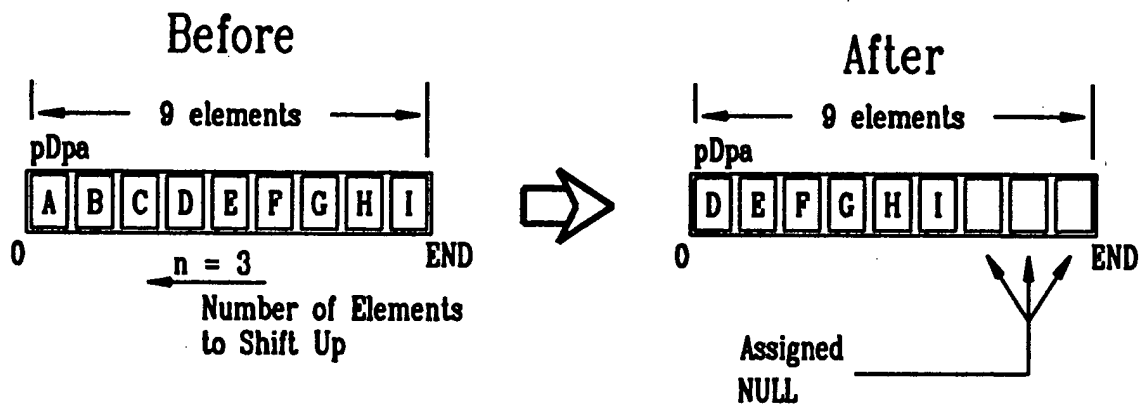
The GenericPointer function may return a value but it is ignored.

## See Also

DpaVisitClient, DpaVisitRange, DpaVisitRegion, DpaVisitSelfAndSuccessors

## DpaShiftUp

### Diagram



# DpaShiftUp

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaShiftUp( pDpa, n )
PDPA      pDpa;
MediumInt n;
```

Public Function

## Purpose

The `DpaShiftUp` function shifts `n` elements up in the *DynamicArray* `pDpa`. The size of the *DynamicArray* `pDpa` is unchanged.

## Parameter - Description

---

<code>pDpa</code>	-	Pointer to a structure of type <i>DynamicArray</i> .
<code>n</code>	-	Number of elements to scroll up in array.

## Return Value

No return value

## Notes

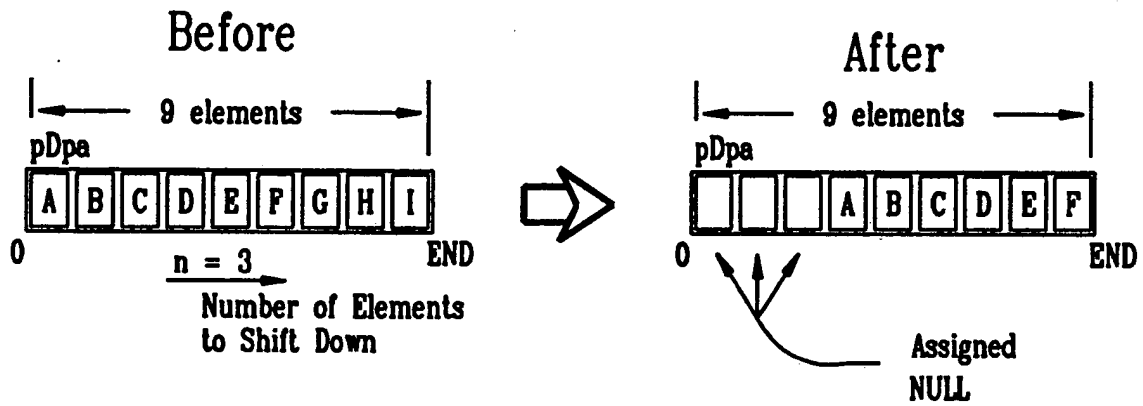
[`N` must be greater than zero and less than the array size.]

## See Also

`DpaAppend`, `DpaDelete`, `DpaExpand`, `DpaShiftDown`

## DpaShiftDown

### Diagram



# DpaShiftDown

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void          DpaShiftDown( pDpa, n )
PDPA          pDpa;
MediumInt     n;
```

Public Function

## Purpose

The DpaShiftDown function shifts n elements down in the *DynamicArray* pDpa. The size of the *DynamicArray* pDpa is unchanged.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
n	-	Number of elements to shift down in array.

## Return Value

No return value

## Notes

[N must be greater than zero and less than the array size.]

## See Also

DpaAppend, DpaDelete, DpaExpand, DpaShiftUp

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaSetSize( pDpa, size )
PDPA      pDpa;
MediumInt size;
```

Public Function

## Purpose

The DpaSetSize function sets the new minimum size of the *DynamicArray* pDpa as a multiple of the increment size.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
size	-	New size of the dynamic array.

## Return Value

No return value

## Notes

If the size of the array is reduced the elements are cut from the end of the array.

If the size of the array is expanded then the additional NULL elements are appended to the end of the array.

[The new size must be greater than or equal to zero.]

## See Also

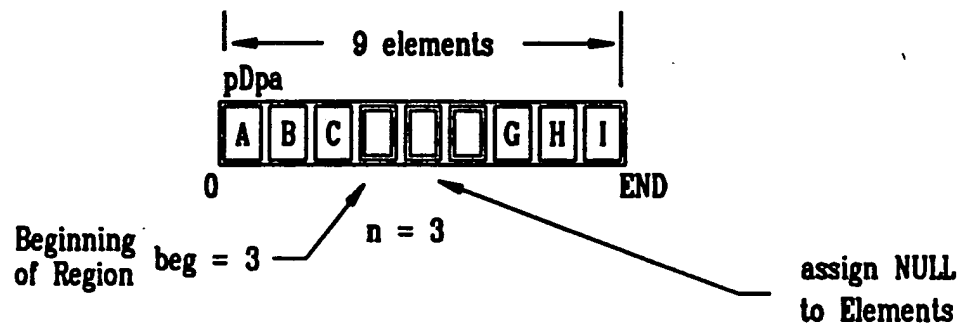
DpaGetSize, DpaInit, DpaNew

## Example

Please refer to class test procedure TSTDPA.C for an example of the use of the DpaSetSize function.

# DpaSetRegionNull

## Diagram



# DpaSetRegionNull

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void          DpaSetRegionNull( pDpa, beg, n )
PDPA          pDpa;
MediumInt     beg;
MediumInt     n;
```

Public Function

## Purpose

The DpaSetRegionNull walks a region of elements, starting with beg for a count of n times, for the *DynamicArray* pDpa and assigns a NULL value to each element visited.

A region of array elements is defined as a beginning element and an count from the beginning element. The count cannot be less than 1. See section on region definition for more details.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
beg	-	Index to start of traverse. [0-END]
n	-	Number of elements to substitute NULL. [1-SIZE]

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

count: SIZE is the number of elements in the array.

## Return Value

No return value

## Notes

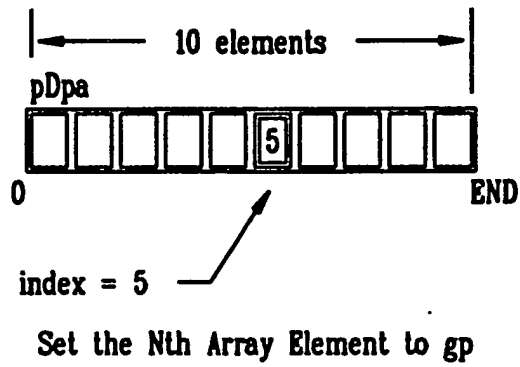
[The index beg must be in the range [0-END].]

[(beg + n) must be less than or equal to the size of the array.]



# DpaSetNth

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void          DpaSetNth( pDpa, index, gp )
PDPA          pDpa;
MediumInt     index;
GenericPtr    gp;
```

Public Function

## Purpose

The DpaSetNth function assigns the GenericPointer gp to the *DynamicArray* pDpa at the element index.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
index	-	Index to element in array.
gp	-	GenericPointer to be assigned to element.

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

## Return Value

No return value

## Notes

[The index index must be in the range [0-END].]

## See Also

DpaGetNth, DpaGetLast

# Dpalnit

---

## See Also

DpaDeInit, DpaDestroy, DpaNew

## Example

Please refer to class test procedure TSTDPA.C for an example of the use of the Dpalnit function.

## Summary

```
#include "cobject.s.h"
#include "dpamac.h"
```

```
Void          Dpalnit( pDpa, s, i )
PDPA          pDpa;
MediumInt     s;
MediumInt     i;
```

Public Function

## Purpose

The Dpalnit function initializes the *DynamicArray* object. The Dpalnit function should be the first function called when using the *DynamicArray* class. The initial size of the array s, and the increment size i for the array are passed. The increment size is used for expanding the array as demanded by appends or pastes. The memory required by the array's initial size is allocated.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
s	-	Initial size of dynamic pointer array.
i	-	Increment size for expansion of array.

## Return Value

No return value

## Notes

The last function to call when using the *DynamicArray* class is DpaDeInit.

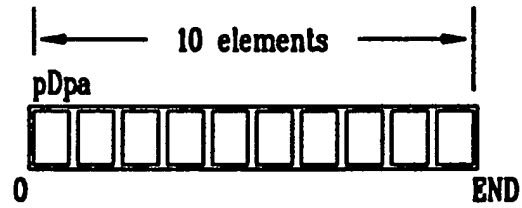
[The increment must be greater than one.]

[The initial size must be greater than or equal to zero.]

# DpaGetSize

---

## Diagram



Get the Size of the Array = 10

# DpaGetSize

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaGetSize( pDpa )
PDPA         pDpa;
```

Public Function

## Purpose

The DpaGetSize function returns the size of the *DynamicArray* pDpa.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
------	---	--

## Return Value

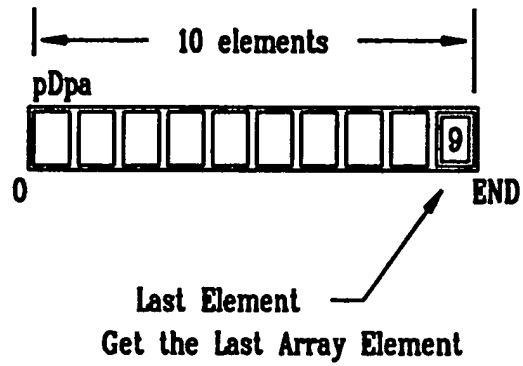
The return value from the DpaGetSize function is the size of the array.

## See Also

DpaInit, DpaNew, DpaSetSize

# DpaGetLast

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
GenericPtr    DpaGetLast( pDpa )
PDPA          pDpa;
```

Public Function

## Purpose

The DpaGetLast function returns the GenericPointer contained in the *DynamicArray* pDpa as the last element.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
------	---	--

## Return Value

The return value from the DpaGetLast function is the GenericPointer contained in the last array element.

## Notes

[The array must have at least one element.]

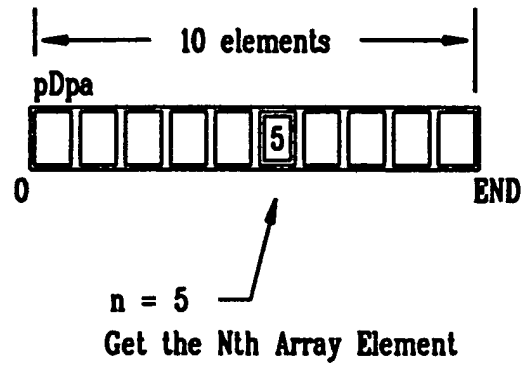
## See Also

DpaGetNth, DpaSetNth



# DpaGetNth

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
GenericPtr    DpaGetNth( pDpa, n )
PDPA          pDpa;
MediumInt     n;
```

Public Function

## Purpose

The DpaGetNth function returns the GenericPointer contained in the *DynamicArray* pDpa at element index.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
index	-	Index to array element. [0-END]

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

## Return Value

The return value from the DpaGetNth function is the GenericPointer contained in *DynamicArray* pDpa at element index.

## Notes

[The index n must be in the range [0-END].]

## See Also

DpaGetLast, DpaSetNth

# DpaFindRangeFwd

## Notes

[The index beg must be in the range [0-END].]

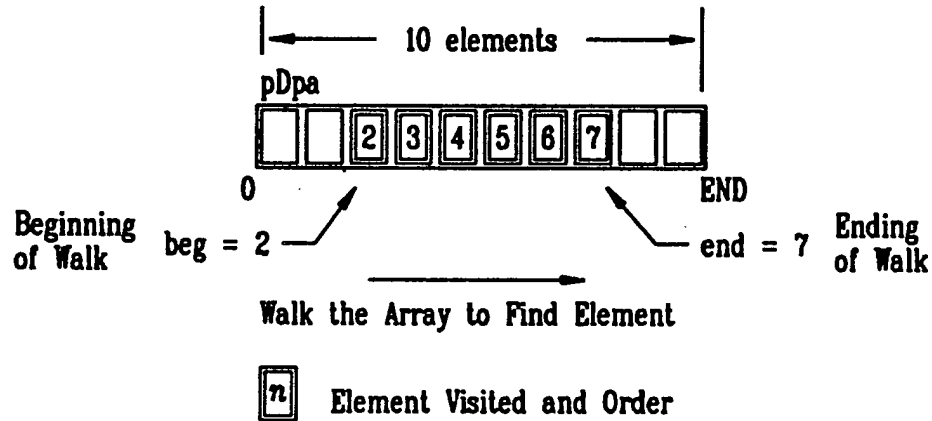
[The index end must be in the range [0-END].]

[beg must be less than or equal to end.]

## See Also

DpaCount, DpaCountRange, DpaFind, DpaFindPtrBwd, DpaFindPtrFwd, DpaFindRangeBwd

## Diagram



# DpaFindRangeFwd

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaFindRangeFwd( pDpa, beg, end, pBlk )
PDPA         pDpa;
MediumInt    beg;
MediumInt    end;
PBLK         pBlk;
```

Public Function

## Purpose

The `DpaFindRangeFwd` function walks a range of array elements, `beg` through `end`, for the *DynamicArray* `pDpa` and calls a `GenericPointer` function for each element visited. The array is walked in a forward direction. The function will terminate when the `GenericPointer` function returns `True` or the end of the range is reached. If a `True` value is returned, the index of the array element is returned.

The *Block* `pBlk` contains the `GenericPointer` function and an optional list of arguments. The function must return a boolean (`True/False`) value.

A range of array elements is defined as a beginning element and an ending element. The beginning element can equal the ending element. See section on range definition for more details.

Parameter	-	Description
-----------	---	-------------

---

<code>pDpa</code>	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
<code>beg</code>	-	Index to beginning of traverse. [0-END]
<code>end</code>	-	Index to end of traverse. [0-END]
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

index: 0 is the first element index in the array and `END` is the last consecutively numbered element index.

## Return Value

The return value from the `DpaFindRangeFwd` function is the index to the first array element for which the `GenericPointer` function returns `True`. Otherwise -1 is returned.

# DpaFindRangeBwd

## Notes

[The index beg must be in the range [0-END].]

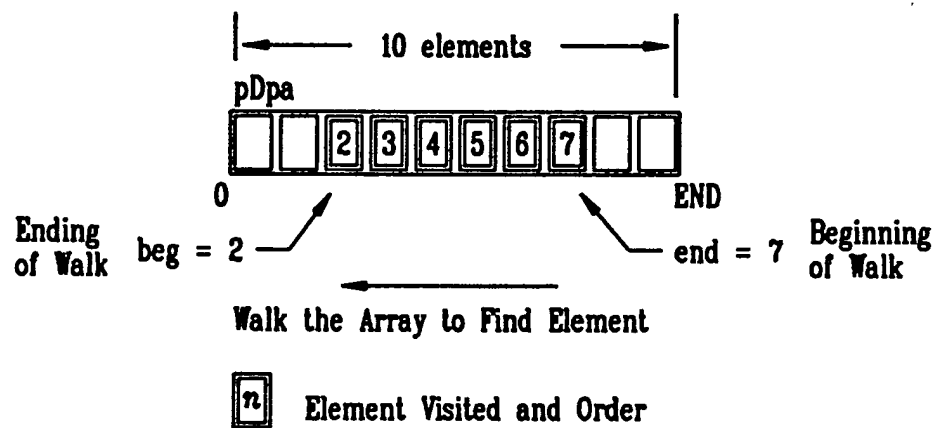
[The index end must be in the range [0-END].]

[beg must be less than or equal to end.]

## See Also

DpaCount, DpaCountRange, DpaFind, DpaFindPtrBwd, DpaFindPtrFwd, DpaFindRangeFwd

## Diagram



# DpaFindRangeBwd

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaFindRangeBwd( pDpa, beg, end, pBlk )
PDPA          pDpa;
MediumInt     beg;
MediumInt     end;
PBLK          pBlk;
```

Public Function

## Purpose

The `DpaFindRangeBwd` function walks a range of array elements, `beg` through `end`, for the *DynamicArray* `pDpa` and calls a `GenericPointer` function for each element visited. The array is walked in a backward direction. The function will terminate when the `GenericPointer` function returns `True` or the end of the range is reached. If a `True` value is returned, the index of the array element is returned.

The *Block* `pBlk` contains the `GenericPointer` function and an optional list of arguments. The function must return a boolean (`True/False`) value.

A range of array elements is defined as a beginning element and an ending element. The beginning element can equal the ending element. See section on range definition for more details.

## Parameter - Description

---

<code>pDpa</code>	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
<code>beg</code>	-	Index to beginning of traverse. [0-END]
<code>end</code>	-	Index to end of traverse. [0-END]
<code>pBlk</code>	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

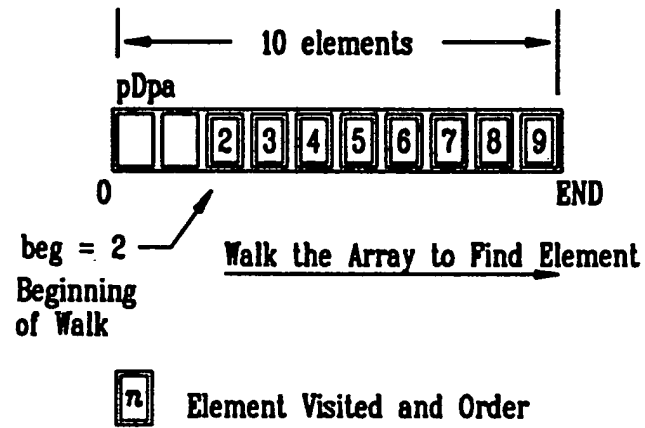
index: 0 is the first element index in the array and END is the last consecutively numbered element index.

## Return Value

The return value from the `DpaFindRangeBwd` function is the index to the first array element for which the `GenericPointer` function returns `True`. Otherwise -1 is returned.

# DpaFindPtrFwd

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaFindPtrFwd( pDpa, beg, gp )
PDPA         pDpa;
MediumInt    beg;
GenericPtr    gp;
```

Public Function

## Purpose

The DpaFindPtrFwd function walks the *DynamicArray* pDpa, starting at beg and ending at the last element of the array, and returns an index to the element for which the GenericPointer gp equals the GenericPointer of the element visited.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
beg	-	Index to start of traverse.
gp	-	GenericPointer to be compared.

## Return Value

The return value from the DpaFindPtrFwd function is the index of the array element for which the GenericPointer of the element equals the GenericPointer gp. If no match is found the function returns -1.

## Notes

[The index beg must be in the range [0-END].]

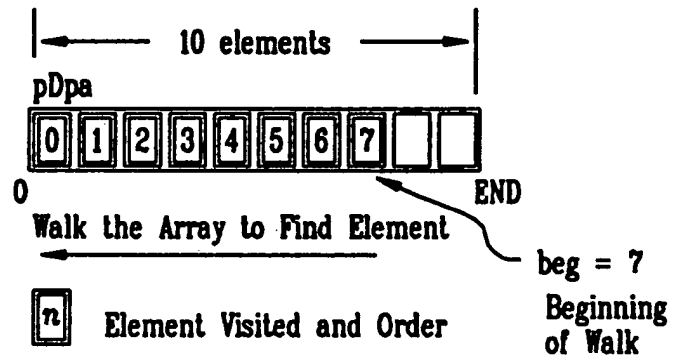
## See Also

DpaCount, DpaCountRange, DpaFind, DpaFindPtrBwd, DpaFindRangeFwd, DpaFindRangeBwd



# DpaFindPtrBwd

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaFindPtrBwd( pDpa, beg, gp )
PDPA         pDpa;
MediumInt    beg;
GenericPtr    gp;
```

Public Function

## Purpose

The DpaFindPtrBwd function walks the *DynamicArray* pDpa, starting at beg and ending at the first element of the array, and returns an index to the element for which the GenericPointer gp equals the GenericPointer of the element visited.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
beg	-	Index to start of traverse.
gp	-	GenericPointer to be compared.

## Return Value

The return value from the DpaFindPtrBwd function is the index of the array element for which the GenericPointer of the element equals the GenericPointer gp. If no match is found the function returns -1.

## Notes

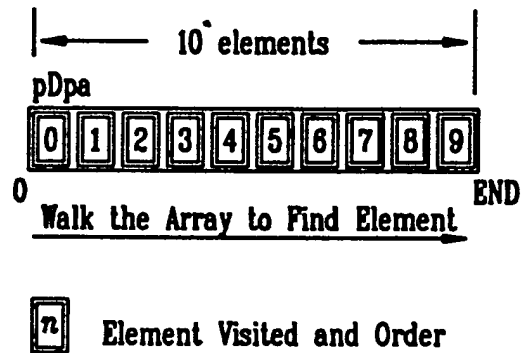
[The index beg must be in the range [0-END].]

## See Also

DpaCount, DpaCountRange, DpaFind, DpaFindPtrFwd, DpaFindRangeFwd, DpaFindRangeBwd

# DpaFind

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaFind( pDpa, pBlk )
PDPA         pDpa;
PBLK         pBlk;
```

Public Function

## Purpose

The DpaFind function walks the *DynamicArray* pDpa and calls a GenericPointer function for each element visited. The array is walked in a forward direction. The function will terminate when the GenericPointer function returns True or the end of the array is reached. If a True value is returned, the index of the array element is returned.

The *Block* pBlk contains the GenericPointer function and an optional list of arguments. The function must return a boolean (True/False) value.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

## Return Value

The return value from the DpaFind function is the index of the first array element for which the GenericPointer function returns True. If no function returns True then the return value is -1.

## See Also

DpaCount, DpaCountRange, DpaFindPtrBwd, DpaFindPtrFwd, DpaFindRangeFwd, DpaFindRangeBwd

# DpaExpand

## Notes (cont)

the caller is required to assign values to the new elements with the DpaSetNth function. The DpaAppend function assigns a GenericPointer value to the appended array element.

At least one element will always be moved down the array.

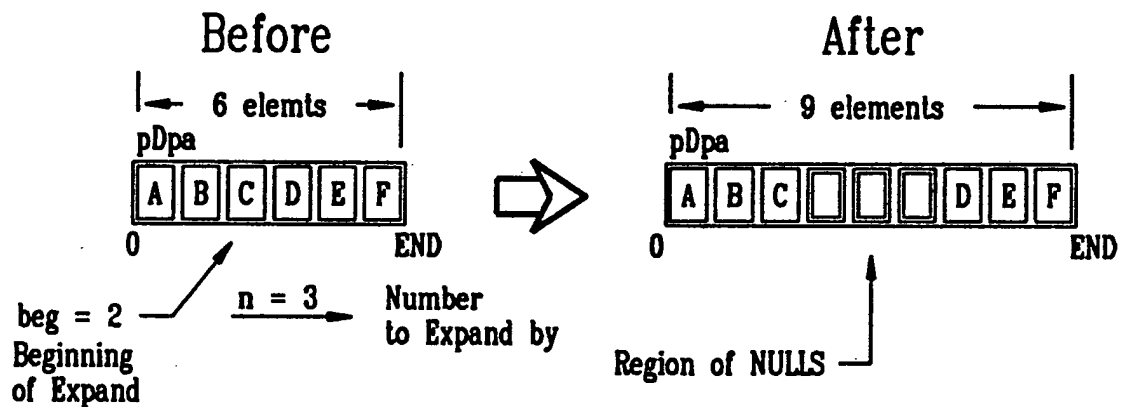
[beg must be in the range [0:DpaGetSize].]

[n must be greater than zero.]

## See Also

DpaAppend, DpaDelete, DpaShiftDown, DpaShiftUp

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void          DpaExpand( pDpa, beg, n )
PDPA          pDpa;
MediumInt     beg;
MediumInt     n;
```

Public Function

## Purpose

The DpaExpand function expands the *DynamicArray* pDpa by inserting NULL values into a region of array elements starting with beg for a count of n times.

A region of array elements is defined as a beginning element and an count from the beginning element. The count cannot be less than 1. See section on region definition for more details.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
beg	-	Index to beginning of the paste. [0-END]
n	-	Number of elements to paste into array. [1-SIZE]

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

count: SIZE is the number of elements in the array.

## Return Value

No return value

## Notes

The size of the array is expanded, if required, to accommodate the additional elements.

This function differs from the DpaAppend function in that

# DpaDestroy

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaDestroy( pDpa )
PDPA      pDpa;
```

Public Function

## Purpose

The DpaDestroy function deallocates the memory used by the object and deinitializes the *DynamicArray* object. The *DynamicArray* pDpa should not be referenced after this function call since its memory will have been deallocated.

## Parameter - Description

---

pDpa - Pointer to a structure of type *DynamicArray*.

## Return Value

No return value

## See Also

DpaDeInit, DpaInit, DpaNew

## Example

Please refer to class test procedure TSTDPA.C for an example of the use of the DpaDestroy function.

## DpaDelete

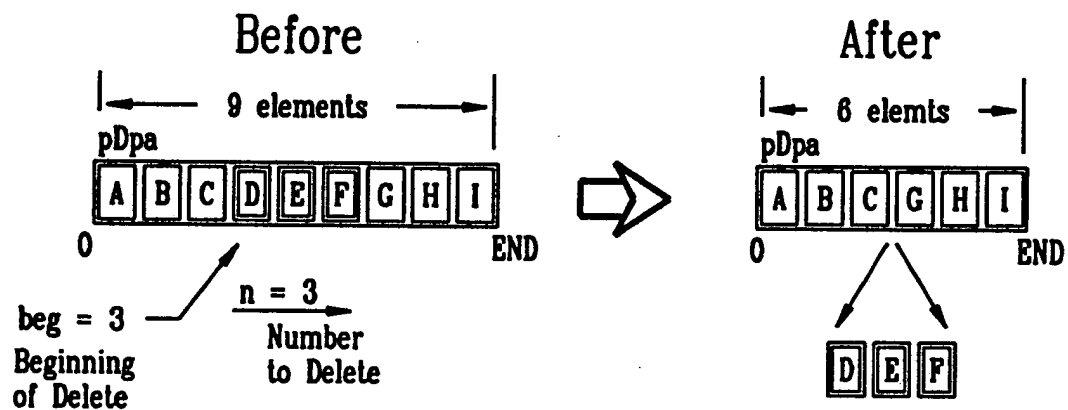
### Notes (cont)

[(beg + n) must be less than or equal to the size of the array.]

### See Also

DpaAppend, DpaExpand, DpaShiftDown, DpaShiftUp

### Diagram





# DpaDelete

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaDelete( pDpa, beg, n )
PDPA      pDpa;
MediumInt beg;
MediumInt n;
```

Public Function

## Purpose

The DpaDelete function deletes a region of array elements, starting with beg for a count of n times, from the *DynamicArray* pDpa.

A region of array elements is defined as a beginning element and an count from the beginning element. The count cannot be less than 1. See section on region definition for more details.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
beg	-	Index to beginning of the delete. [0-END]
n	-	Number of elements to delete from array. [1-SIZE]

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

count: SIZE is the number of elements in the array.

## Return Value

No return value

## Notes

The array size is reduced by the number of elements that are deleted.

[The index beg must be in the range [0-END].]

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
Void      DpaDeInit( pDpa )
PDPA      pDpa;
```

Public Function

## Purpose

The DpaDeInit function deinitializes the *DynamicArray* object. The DpaDeInit function should be the last function called when using the *DynamicArray* class. The memory used by the array is deallocated.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
------	---	--

## Return Value

No return value

## Notes

The first function to call when using the *DynamicArray* class is DpaInit.

## See Also

DpaDestroy, DpaInit, DpaNew

## Example

Please refer to class test procedure TSTDPA.C for an example of the use of the DpaDeInit function.

# DpaCountRange

## Notes (cont)

[The index beg must be in the range [0-END].]

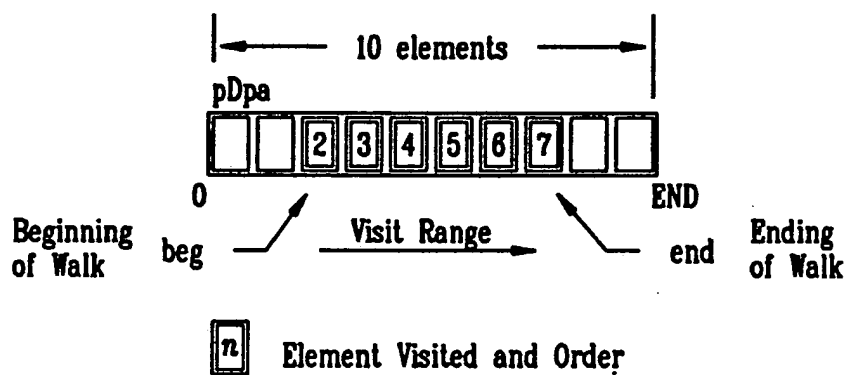
[The index end must be in the range [0-END].]

[beg must be less than or equal to end.]

## See Also

DpaCount, DpaFind, DpaFindRangeFwd, DpaFindRangeBwd

## Diagram



# DpaCountRange

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaCountRange( pDpa, beg, end, pBlk )
PDPA          pDpa;
MediumInt     beg;
MediumInt     end;
PBLK          pBlk;
```

Public Function

## Purpose

The DpaCountRange function returns the number of times that a GenericPointer function returns a True value. The *DynamicArray* pDpa is walked for a range of elements beg through end in a forward direction with the GenericPointer function being called once for each element visited.

The *Block* pBlk contains the GenericPointer function and an optional list of arguments. The function must return a boolean (True/False) value.

A range of array elements is defined as a beginning element and an ending element. The beginning element can equal the ending element. See section on range definition for more details.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
beg	-	Index to beginning of traverse. [0-END]
end	-	Index to end of traverse. [0-END]
pBlk	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

index: 0 is the first element index in the array and END is the last consecutively numbered element index.

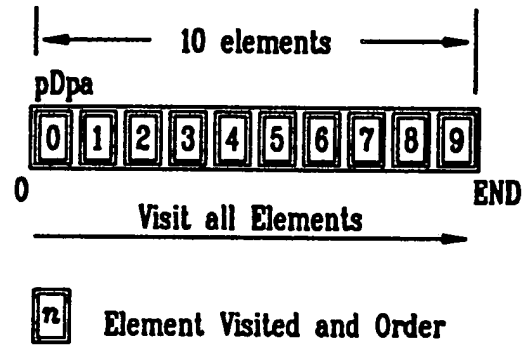
## Return Value

The return value from the DpaCountRange function is the number of times the GenericPointer function returns True for a range of array elements.

## Notes

# DpaCount

## Diagram



## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaCount( pDpa, pBlk )
PDPA          pDpa;
PBLK          pBlk;
```

Public Function

## Purpose

The DpaCount function returns the number of times that a GenericPointer function returns a True value. The entire *DynamicArray* pDpa is walked in a forward direction with the GenericPointer function being called once for each element visited.

The *Block* pBlk contains the GenericPointer function and an optional list of arguments. The function must return a boolean (True/False) value.

Parameter	-	Description
-----------	---	-------------

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . This is the array to walk.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the function to call and any optional parameters to be sent to the function.

## Return Value

The return value from the DpaCount function is the number of times the GenericPointer function returns True.

## See Also

DpaCountRange, DpaFind, DpaFindRangeFwd, DpaFindRangeBwd

# DpaClear

---

## Summary

```
#include "cobjects.h"  
#include "dpamac.h"
```

```
Void      DpaClear( pDpa )  
PDPA      pDpa;
```

Public Function

## Purpose

The DpaClear function sets the array to its initial state and sets the size of the array to 0.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> . The array being cleared.
------	---	---

## Return Value

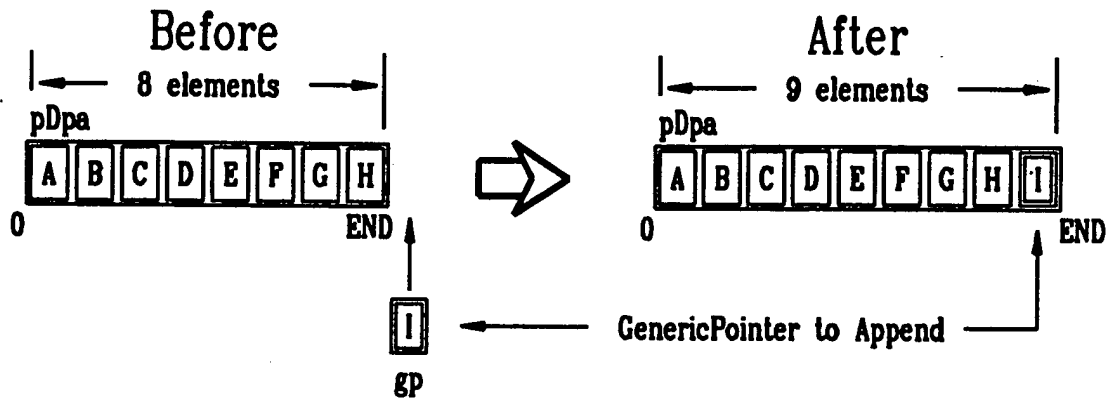
No return value

## Example

Please refer to class test procedure TSTDPA.C for an example of the use of the DpaClear function.

# DpaAppend

Diagram





# DpaAppend

---

## Summary

```
#include "cobjects.h"
#include "dpamac.h"
```

```
MediumInt    DpaAppend( pDpa, gp )
PDPA         pDpa;
GenericPtr    gp;
```

Public Function

## Purpose

The DpaAppend function appends the GenericPointer gp to be last in the *DynamicArray* pDpa.

## Parameter - Description

---

pDpa	-	Pointer to a structure of type <i>DynamicArray</i> .
gp	-	GenericPointer to be appended to the array.

## Return Value

The return value from the DpaAppend function is the size of the array after the GenericPointer gp has been appended.

## Notes

This function differs from the DpaExpand function in that it assigns a GenericPointer to the new array element whereas DpaExpand opens up the array and the caller is required to assign values to the new elements with the DpaSetNth function.

The array is expanded if needed.

## See Also

DpaDelete, DpaExpand, DpaShiftDown, DpaShiftUp

## Class Reference for *DynamicArray*

**Structure Name:**      DynamicArray

**Abbreviation:**        Dpa

**Class Type:**            Primitive

# DllSendDestroy

---

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
Void      DllSendDestroy( pDll )  
PDLL      pDll;
```

Public Function

## Purpose

The DllSendDestroy function sends a message to the client of the *List* pDll asking it to destroy the *List*. The *List* client function will receive this message and should destroy the list element. This message function should be included in the *List* client message array.

## Return Value

No return value

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllSendDestroy function.

## DllPasteRangeLast

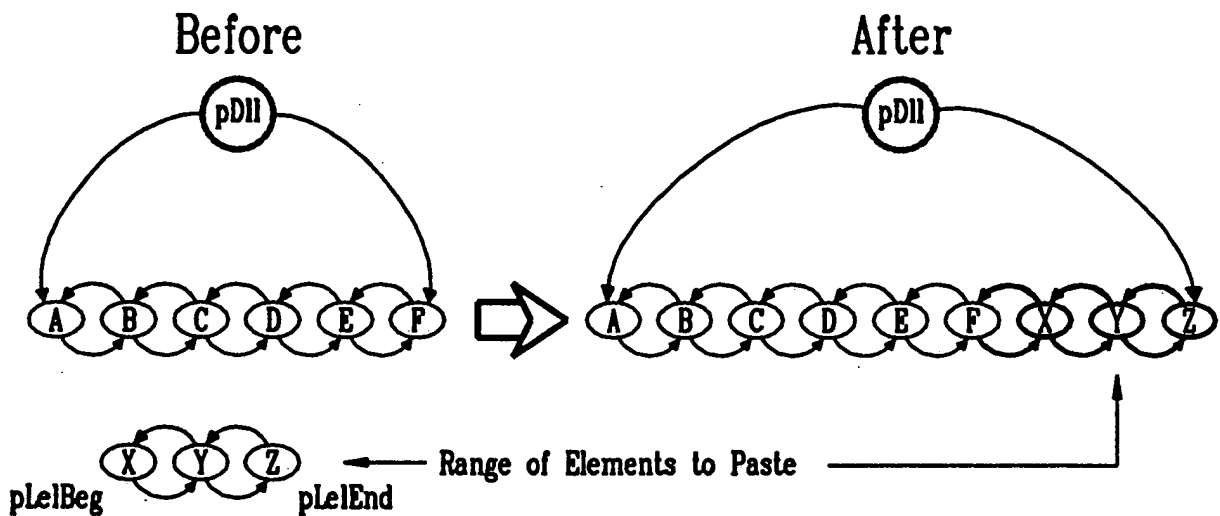
### Notes (cont)

pLel must be linked to the list.

### See Also

DllClear, DllCut, DllCutRange, DllPasteRangeAfter, DllPasteRangeBefore, DllPasteRangeFirst

### Diagram



# DllPasteRangeLast

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllPasteRangeLast( pDll, pLelBeg, pLelEnd )
PDLL          pDll;
PLEL          pLelBeg;
PLEL          pLelEnd;
```

Public Function

A macro is available for this function

## Purpose

The DllPasteRangeLast function links a range of *ListElements* pLelBeg through pLelEnd to the *List* pDll as the last list elements of the list.

The list element range must have been previously cut from a list.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *List* class section on range definition for more details.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list to modify.
pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to link to the list.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to link to the list.

## Return Value

No return value

## Notes

pLelBeg through pLelEnd cannot currently belong to a list.

If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.

The pLelBeg must have a set of successors one of which is pLelEnd.

## DllPasteRangeFirst

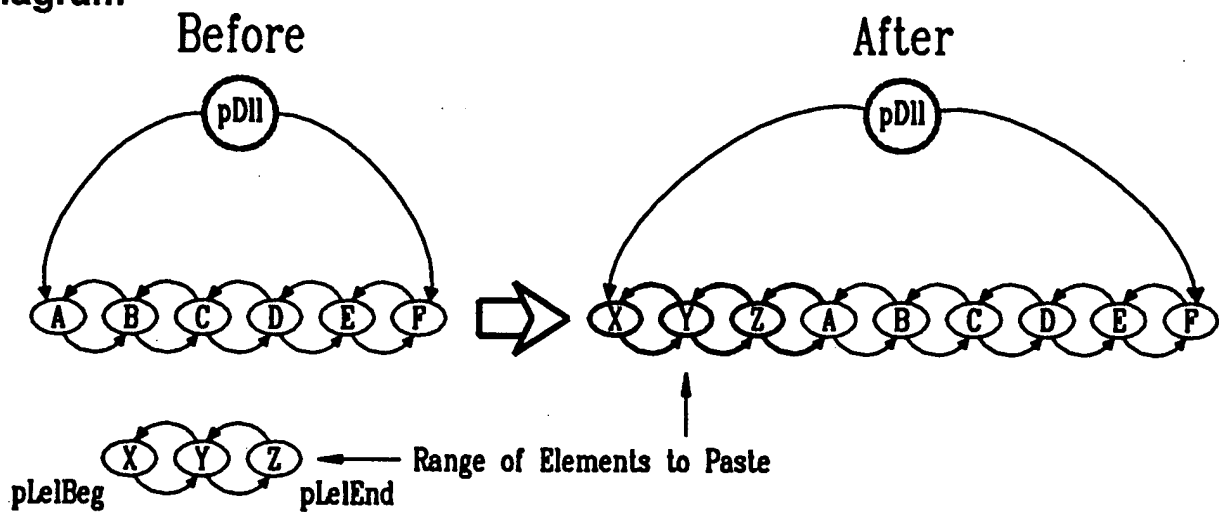
### Notes (cont)

pLel must be linked to the list.

### See Also

DllClear, DllCut, DllCutRange, DllPasteRangeAfter, DllPasteRangeBefore, DllPasteRangeLast

### Diagram



# DllPasteRangeFirst

---

## Summary

```
#include "cobjects.h"
```

```
#include "dllmac.h"
```

```
Void      DllPasteRangeFirst( pDll, pLelBeg, pLelEnd )
PDLL      pDll;
PLEL      pLelBeg;
PLEL      pLelEnd;
```

Public Function

A macro is available for this function

## Purpose

The DllPasteRangeFirst function links a range of *ListElements* pLelBeg through pLelEnd to the *List* pDll as the first list elements of the list.

The list element range must have been previously cut from a list.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *List* class section on range definition for more details.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list to modify.
pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to link to the list.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to link to the list.

## Return Value

No return value

## Notes

pLelBeg through pLelEnd cannot currently belong to a list.

If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.

The pLelBeg must have a set of successors one of which is pLelEnd.

## DllPasteRangeBefore

### Notes (cont)

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

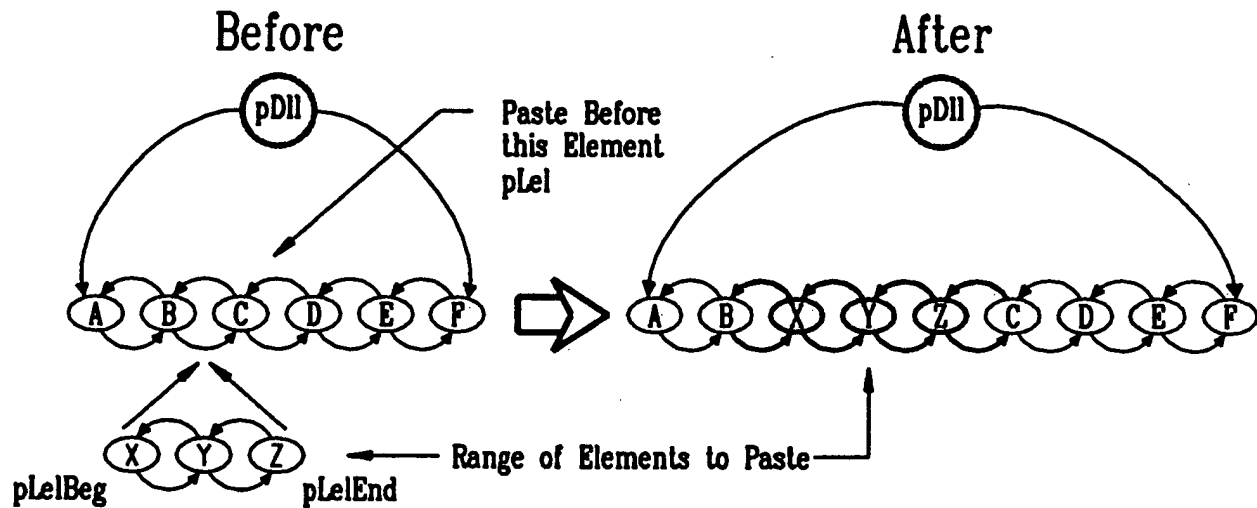
[pLelBeg must not already be in a list.]

[pLel must be in the list pDll.]

### See Also

DllClear, DllCut, DllCutRange, DllPasteRangeAfter, DllPasteRangeFirst, DllPasteRangeLast

### Diagram





# DllPasteRangeBefore

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllPasteRangeBefore( pDll, pLel, pLelBeg, pLelEnd )
PDLL          pDll;
PLEL          pLel;
PLEL          pLelBeg;
PLEL          pLelEnd;
```

Public Function

## Purpose

The DllPasteRangeBefore function links a range of *ListElements* pLelBeg through pLelEnd to the *List* pDll as the preceding list elements of the *ListElement* pLel. If the *ListElement* pLel is NULL then the *ListElements* pLelBeg through pLelEnd are linked to the *List* pDll as the first elements.

The list element range must have been previously cut from a list.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *List* class section on range definition for more details.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list to modify.
pLel	-	Pointer to a structure of type <i>ListElement</i> . pLelBeg through pLelEnd will be linked as the predecessors of this list element.
pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to link to the list.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to link to the list.

## Return Value

No return value

## Notes

The pLelBeg must have a set of successors one of which is pLelEnd.

## DllPasteRangeAfter

### Notes (cont)

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

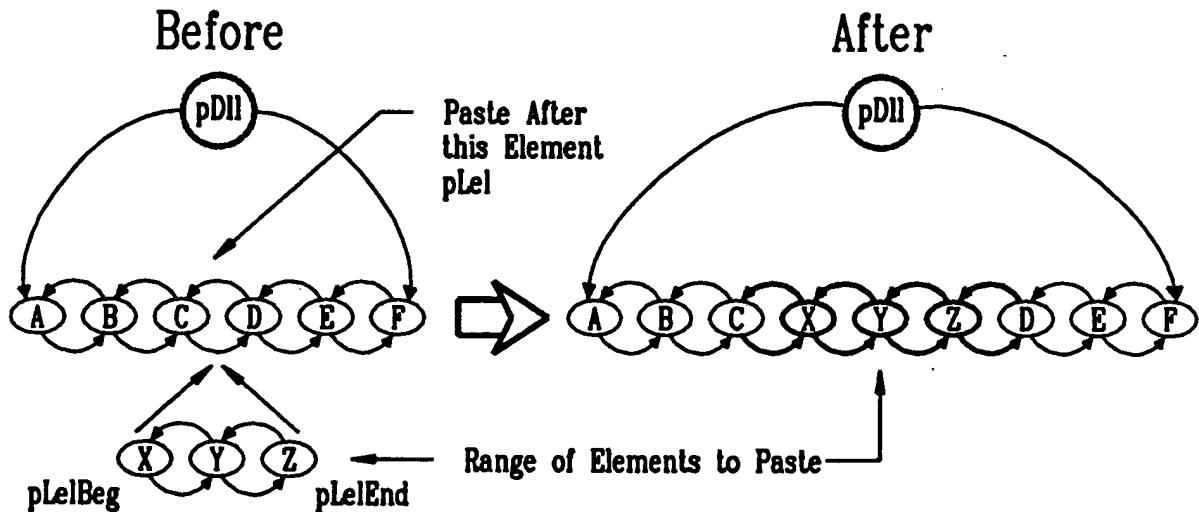
[pLelBeg must not already be in a list.]

[pLel must be in the list pDll.]

### See Also

DllClear, DllCut, DllCutRange, DllPasteRangeBefore, DllPasteRangeFirst, DllPasteRangeLast

### Diagram



# DllPasteRangeAfter

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void      DllPasteRangeAfter( pDll, pLel, pLelBeg, pLelEnd )
PDLL      pDll;
PLEL      pLel;
PLEL      pLelBeg;
PLEL      pLelEnd;
```

Public Function

## Purpose

The DllPasteRangeAfter function links a range of *ListElements* pLelBeg through pLelEnd to the *List* pDll as the succeeding list elements of the *ListElement* pLel. If the *ListElement* pLel is NULL then the *ListElements* pLelBeg through pLelEnd are linked to the *List* pDll as the last elements.

The list element range must have been previously cut from a list.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *List* class section on range definition for more details.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list to modify.
pLel	-	Pointer to a structure of type <i>ListElement</i> . pLelBeg through pLelEnd will be linked as the successors of this list element.
pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the first list element to link to the list.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the last list element to link to the list.

## Return Value

No return value

## Notes

The pLelBeg must have a set of successors one of which is pLelEnd.

# DllNotifyPasteRange

---

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
Void          DllNotifyPasteRange( pDll, pLelBeg, pLelEnd )  
PDLL          pDll;  
PLEL          pLelBeg;  
PLEL          pLelEnd;
```

Friend Function

## Purpose

The DllNotifyPasteRange is called by the *ListElement* class to notify the *List* that elements were pasted.

## Return Value

No return value

## Notes

This is a function is only to be used by the *ListElement* class.

## See Also

DllNotifyCutRange

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllNotifyPasteRange function.

# DllNotifyCutRange

---

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
Void      DllNotifyCutRange( pDll, pLelBeg, pLelEnd )  
PDLL      pDll;  
PLEL      pLelBeg;  
PLEL      pLelEnd;
```

Friend Function

## Purpose

The DllNotifyCutRange is called by the *ListElement* class to notify the *List* that elements were cut.

## Return Value

No return value

## Notes

This is a function is only to be used by the *ListElement* class.

## See Also

DllNotifyPasteRange

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllNotifyCutRange function.

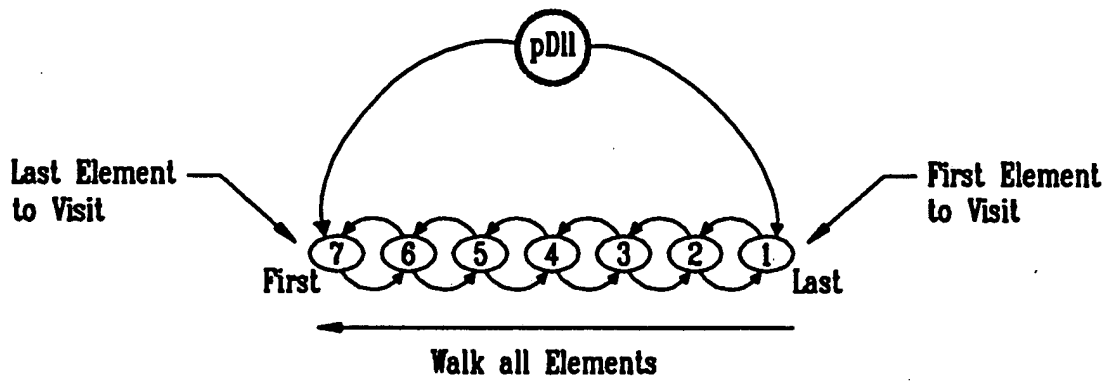
## DllLelClientVisitFwd

---

### See Also

DllLelClientVisitBwd

### Diagram



# DllLelClientVisitFwd

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllLelClientVisitFwd( pDll, offset, pBlk )
PDLL          pDll;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The DllLelClientVisitFwd function walks the entire *List* pDll and calls a *ListElement* client function for each list element. The list is walked in a forward direction.

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list to walk.
offset	-	The distance in bytes between a <i>ListElement</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

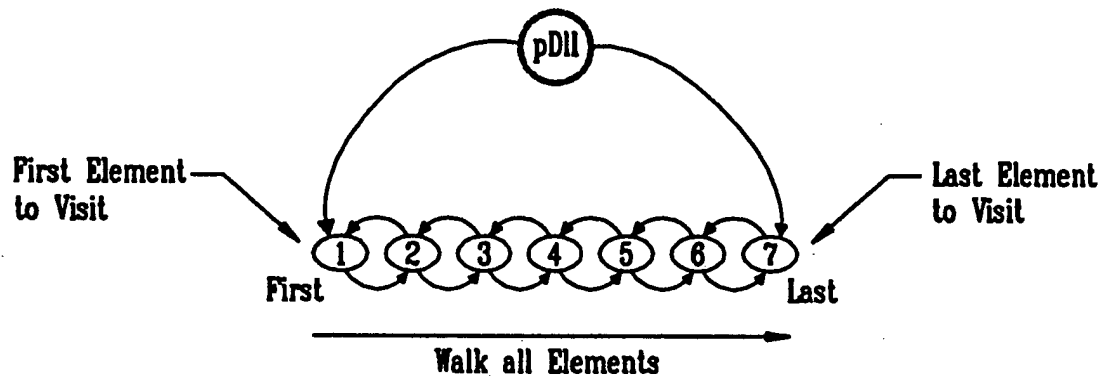
## DllLeIClientVisitBwd

---

### See Also

DllLeIClientVisitFwd

### Diagram





# DllLelClientVisitBwd

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllLelClientVisitBwd( pDll, offset, pBlk )
PDLL          pDll;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The DllLelClientVisitBwd function walks the entire *List* pDll and calls a *ListElement* client function for each list element. The list is walked in a backward direction (last to first).

The *Block* pBlk contains the client function and an optional list of arguments.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list to walk.
offset	-	The distance in bytes between a <i>ListElement</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

No return value

## Notes

The client function may return a value but it is ignored.

## DllLelClientLast

---

### Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
POBJ      DllLelClientLast( pDll, offset )
PDLL      pDll;
```

Public Function

A macro is available for this function

### Purpose

The DllLelClientLast function returns the client pointer of the last *ListElement* in the *List* pDll.

### Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> .
offset	-	The distance in bytes between a <i>ListElement</i> and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

### Return Value

The return value from the DllLelClientLast function is a pointer to the client of the last list element in the *List* pDll. If the list is empty the function will return NULL.

### See Also

DllLelClientFirst

### Example

Please refer to class test procedure TSTDLL.C,TSTLEL.C for an example of the use of the DllLelClientLast function.

## **DllLeIClientGetNth**

---

### **Example**

Please refer to class test procedure TSTDLL.C,TSTLEL.C for an example of the use of the DllLeIClientGetNth function.

## DllLelClientGetNth

---

### Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
POBJ      DllLelClientGetNth( pDll, offset, index )
PDLL      pDll;
MediumInt offset;
MediumInt index;
```

Public Function

### Purpose

The DllLelClientGetNth function returns the client pointer of the Nth *ListElement* in the *List* pDll.

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> .
offset	-	The distance in bytes between a <i>ListElement</i> and it's client pointer. The value must be 0 or negative.
index	-	Index to list element in list. [0-END]

offset: See the *Class Data Structures* reference guide for details on using offsets.

index: 0 is the first list element index in the list and END is the last consecutively numbered list element index.

### Return Value

The return value from the DllLelClientGetNth function is the client pointer of the indexed list element in the *List* pDll. If the list is empty the function will return NULL.

### See Also

DllGetNth

# DllLelClientFirst

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
POBJ      DllLelClientFirst( pDll, offset )
PDLL      pDll;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The DllLelClientFirst returns the client pointer of the first *ListElement* in the *List* pDll.

---

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> .
offset	-	The distance in bytes between a <i>ListElement</i> and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the DllLelClientFirst function is a pointer to the client of the first list element in the *List* pDll. If the list is empty the function will return NULL.

## See Also

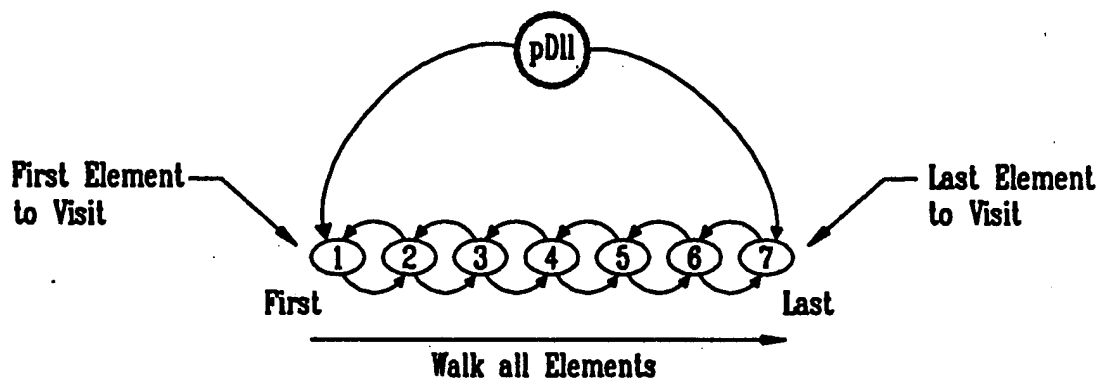
DllLelClientLast

## Example

Please refer to class test procedure TSTDLL.C,TSTLEL.C for an example of the use of the DllLelClientFirst function.

## DIILeIClientFind

Diagram



# DllLelClientFind

---

## Summary

```
#include "cobjects.h"
```

```
#include "dllmac.h"
```

```
POBJ      DllLelClientFind( pDll, offset, pBlk )
PDLL      pDll;
MediumInt offset;
PBLK      pBlk;
```

Public Function

## Purpose

The DllLelClientFind function walks the *List* pDll and calls a *ListElement* client function for each list element. The list is walked in a forward direction. The function terminates when the client function returns True or the end of the list is reached. If a True value is returned, the client pointer of the list element is returned, otherwise NULL is returned.

The *Block* pBlk contains the client function and an optional list of arguments. The client function must return a boolean (True/False) value.

## Parameter - Description

---

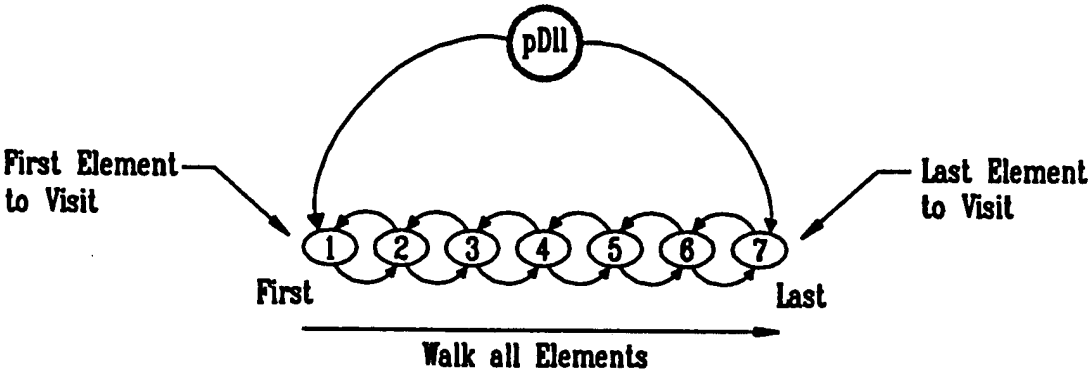
pDll	-	Pointer to a structure of type <i>List</i> . This is the list to walk.
offset	-	The distance in bytes between a <i>ListElement</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the DllLelClientFind function is a pointer to the client of the first list element for which the client function returns True. Otherwise NULL is returned.

Diagram





# DllLelClientCount

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
MediumInt    DllLelClientCount( pDll, offset, pBlk )
PDLL          pDll;
MediumInt     offset;
PBLK          pBlk;
```

Public Function

## Purpose

The DllLelClientCount function returns the number of times that a *ListElement* client function returns non-zero. The entire *List* pDll is walked in a forward direction with the client function being called once for each list element visited.

The *Block* pBlk contains the client function and an optional list of arguments. The client function must return a MediumInt value.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list to walk.
offset	-	The distance in bytes between a <i>ListElement</i> and it's client pointer. The value must be 0 or negative.
pBlk	-	Pointer to structure of type <i>Block</i> which contains the client function to call and any optional parameters to be sent to the function.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the DllLelClientCount function is the number of times the *ListElement* client function returns non-zero.

## DllIsEmpty

---

### Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
Bool      DllIsEmpty( pDll )  
PDLL      pDll;
```

Public Function

A macro is available for this function

### Purpose

The DllIsEmpty function determines if the *List* pDll has any elements.

### Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

### Return Value

The return value from the DllIsEmpty is True if the *List* pDll contains no list elements or False if it does.

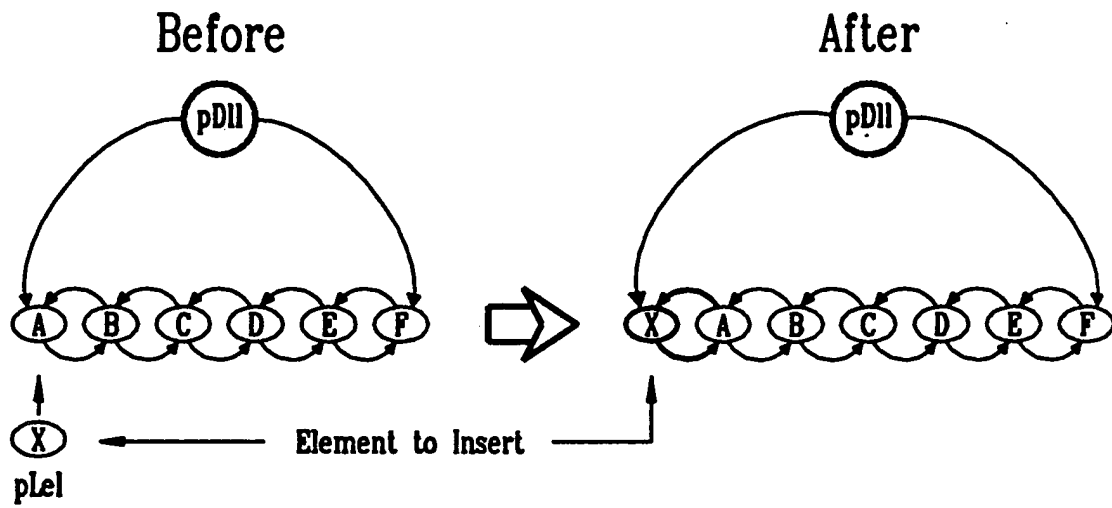
### Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllIsEmpty function.

## DllInsertFirst

---

### Diagram



# DllInsertFirst

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllInsertFirst( pDll, pLel )
PDLL          pDll;
PLEL          pLel;
```

Public Function

A macro is available for this function

## Purpose

The DllInsertFirst function links the *ListElement* pLel as the first list element of the *List* pDll.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . The list being modified.
pLel	-	Pointer to a structure of type <i>ListElement</i> . The list element will be the first in the list.

## Return Value

No return value

## Notes

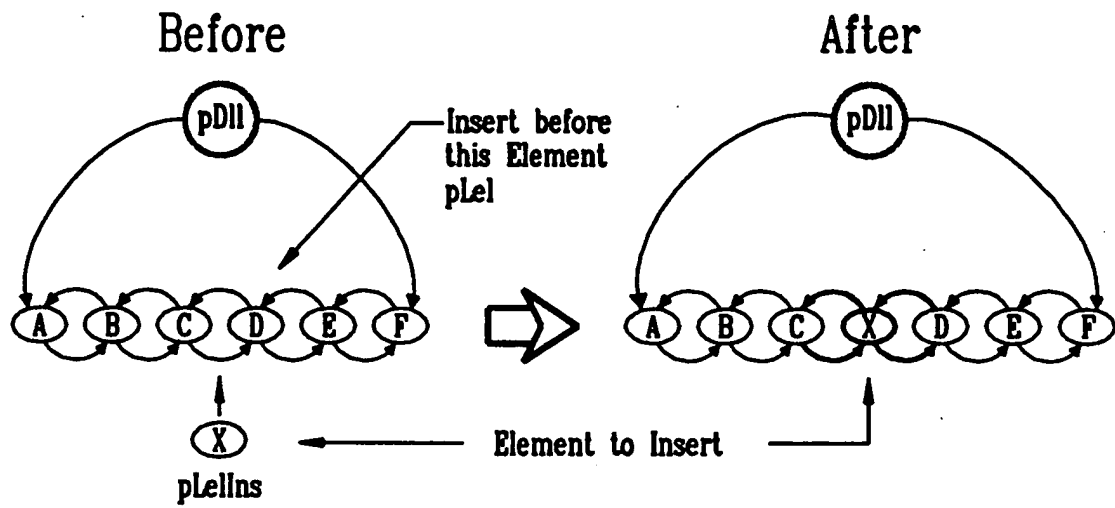
pLel cannot already belong to a list.

## See Also

DllAppend, DllAppendLast, DllInsert

# DllInsert

## Diagram



## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllInsert( pDll, pLel, pLelIns )
PDLL          pDll;
PLEL          pLel;
PLEL          pLelIns;
```

Public Function

## Purpose

The DllInsert function links the *ListElement* pLelIns, preceding the *ListElement* pLel, to the *List* pDll.

If the *ListElement* pLel is NULL then pLelIns is inserted to be first in the *List* pDll.

---

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> . The list being modified.
pLel	-	Pointer to a structure of type <i>ListElement</i> . pLelIns will be linked as the predecessor to this list element.
pLelIns	-	Pointer to a structure of type <i>ListElement</i> . This is the list element to link to the list.

## Return Value

No return value

## Notes

pLelIns cannot already belong to a list.

pLel, if not NULL, must already be linked to pDll.

## See Also

DllAppend, DllAppendLast, DllInsertFirst

# DllInit

---

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
Void      DllInit( pDll )  
PDLL      pDll;
```

Public Function

## Purpose

The DllInit function initializes the *List* object. The DllInit function should be the first function called when using the *List* class.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

## Return Value

No return value

## Notes

The last function to call when using the *List* class is DllDeInit.

## See Also

DllDeInit, DllDestroy

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllInit function.

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
MediumInt    DllGetSize( pDll )  
PDLL         pDll;
```

Public Function

## Purpose

The DllGetSize function returns the number of list elements in the *List* pDll.

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

## Return Value

The return value from the DllGetSize function is the the number of list elements in the *List* pDll.

## Example

Please refer to class test procedure TSTDLL.C,TSTLEL.C for an example of the use of the DllGetSize function.



# DllGetNth

---

## Summary

#include "cobjects.h"

#include "dllmac.h"

PLEL            DllGetNth( pDll, index )  
PDLL            pDll;  
MediumInt       index;

Private Function

## Purpose

The DllGetNth function returns a pointer to the Nth *ListElement* in the *List* pDll.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> .
index	-	Index to list element in list. [0-END]

index: 0 is the first list element index in the list and END is the last consecutively numbered list element index.

## Return Value

The return value from the DllGetNth function is a pointer to a structure of type *ListElement*. The list element is the indexed member of the list.

## See Also

DllGetFirst, DllGetLast

## Example

Please refer to class test procedure TSTDLL.C,TSTLEL.C for an example of the use of the DllGetNth function.

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
PLEL      DllGetLast( pDll )
PDLL      pDll;
```

Private Function

A macro is available for this function

## Purpose

The DllGetLast function returns a pointer to the last *ListElement* in the *List* pDll or NULL if it is empty.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

## Return Value

The return value from the DllGetLast function is a pointer to a structure of type *ListElement*. The list element is the last list element in the *List* pDll or NULL if the list is empty.

## See Also

DllGetFirst, DllGetNth

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllGetLast function.

# DllGetFirst

---

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
PLEL      DllGetFirst( pDll )  
PDLL      pDll;
```

Private Function

A macro is available for this function

## Purpose

The DllGetFirst function returns a pointer to the first *ListElement* in the *List* pDll or NULL if it is empty.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

## Return Value

The return value from the DllGetFirst function is a pointer to a structure of type *ListElement*. The list element is the first in the *List* pDll or NULL if the list is empty.

## See Also

DllGetLast, DllGetNth

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllGetFirst function.

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
POBJ      DllGetClient( pDll, offset )
PDLL      pDll;
MediumInt offset;
```

Public Function

A macro is available for this function

## Purpose

The DllGetClient function returns the client pointer of the *List* pDll.

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> .
offset	-	The distance in bytes between the <i>List</i> pDll and it's client pointer. The value must be 0 or negative.

offset: See the *Class Data Structures* reference guide for details on using offsets.

## Return Value

The return value from the DllGetClient function is the client pointer of the *List* pDll.

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllGetClient function.

# DllDestroy

---

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
Void          DllDestroy( pDll )  
PDLL          pDll;
```

Public Function

## Purpose

The DllDestroy function deallocates the memory used by the object and deinitializes the *List* object. The *List* pDll should not be referenced after this function call since its memory will have been deallocated.

Any elements in the list will be unlinked before destroying the object.

## Parameter - Description

---

pDll - Pointer to a structure of type *List*.

## Return Value

No return value

## Notes

[pDll must not have a sub-object.]

## See Also

DllDeInit, DllInit

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllDestroy function.

## Summary

```
#include "cobjects.h"  
#include "dllmac.h"
```

```
Void          DllDeInit( pDll )  
PDLL          pDll;
```

Public Function

## Purpose

The DllDeInit function deinitializes the *List* object. The DllDeInit function should be the last function called when using the *List* class.

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

## Return Value

No return value

## Notes

The first function to call when using the *List* class is DllInit.

[pDll cannot have any list elements.]

## See Also

DllDestroy, DllInit, DllClear

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllDeInit function.

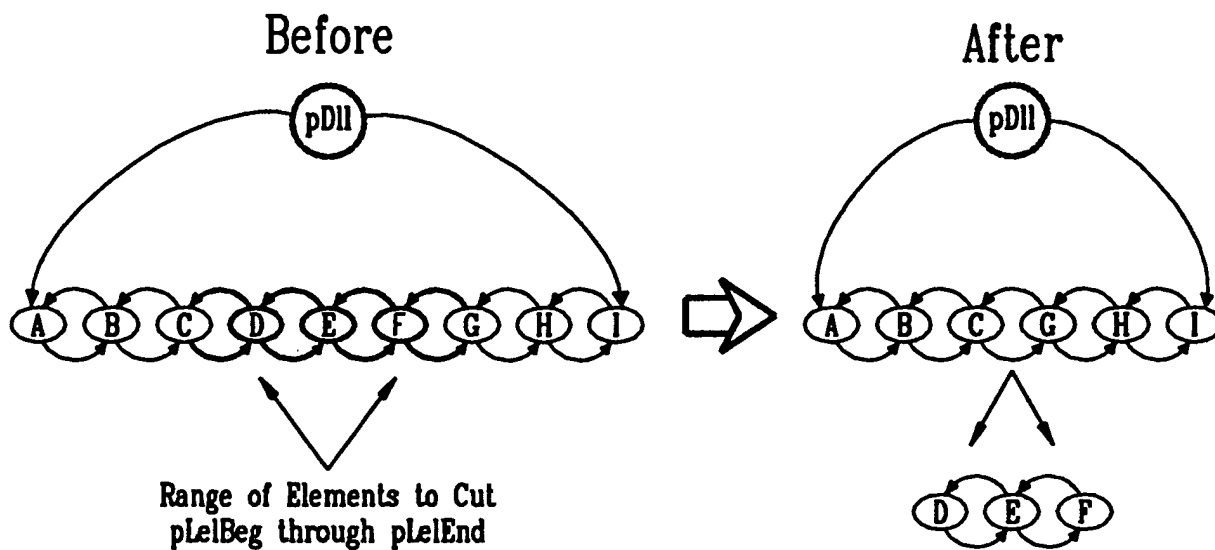
# DllCutRange

---

## See Also

DllClear, DllCut, DllPasteRangeAfter, DllPasteRangeBefore, DllPasteRangeFirst, DllPasteRangeLast

## Diagram



## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllCutRange( pDll, pLelBeg, pLelEnd )
PDLL          pDll;
PLEL          pLelBeg;
PLEL          pLelEnd;
```

Public Function

## Purpose

The DllCutRange function unlinks a range of *ListElements* pLelBeg through pLelEnd from the *List* pDll. The list element preceding pLelBeg will be linked to the successor list element of pLelEnd.

A range of list elements is defined as a beginning list element and an ending list element. The beginning element can equal the ending element. See *List* class section on range definition for more details.

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list being cut.
pLelBeg	-	Pointer to a structure of type <i>ListElement</i> . This is the beginning list element to cut.
pLelEnd	-	Pointer to a structure of type <i>ListElement</i> . This is the ending list element to cut.

## Return Value

No return value

## Notes

The list elements that are cut remain linked but do not belong to any list. All elements (inclusive) between pLelBeg and pLelEnd are unlinked from pDll.

[If pLelBeg does not equal pLelEnd then pLelBeg must precede pLelEnd.]

[pLelBeg must be in the list pDll.]



# DllCutChildren

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void          DllCutChildren( pDll )
PDLL          pDll;
```

Public Function

## Purpose

The DllCutChildren function cuts all list elements (if any) from the *List* pDll. The list will be in the same state as it was after being initialized.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . The list being cleared.
------	---	--

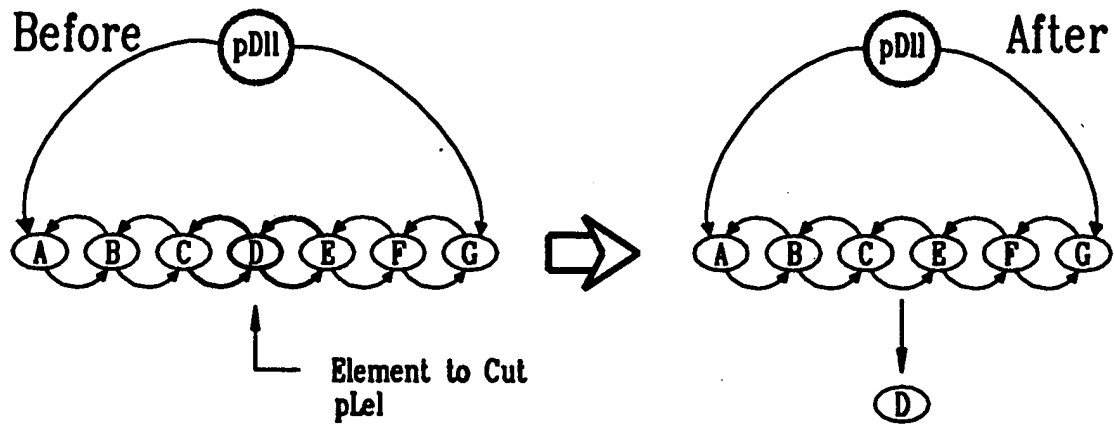
## Return Value

No return value

## Example

Please refer to class test procedure TSTDLL.C,TSTLEL.C for an example of the use of the DllCutChildren function.

## Diagram



# DllCut

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void      DllCut( pDll, pLel )
PDLL      pDll;
PLEL      pLel;
```

Public Function

## Purpose

The DllCut function unlinks the *ListElement* pLel from from the *List* pDll.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . This is the list being modified.
pLel	-	Pointer to a structure of type <i>ListElement</i> . This is the list element to cut.

## Return Value

No return value

## Notes

[pLel must be in the list pDll.]

## See Also

DllClear, DllCutRange, DllPasteRangeAfter, DllPasteRangeBefore, DllPasteRangeFirst, DllPasteRangeLast

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void      DllClear( pDll )
PDLL      pDll;
```

Public Function

## Purpose

The DllClear function cuts all list elements (if any) from the *List* pDll. The list will be in the same state as it was after being initialized.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

## Return Value

No return value

## Example

Please refer to class test procedure TSTDLL.C, TSTLEL.C for an example of the use of the DllClear function.

# DllAsObj

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
POBJ      DllAsObj( pDll )
PDLL      pDll;
```

Private Function

A macro is available for this function

## Purpose

The DllAsObj function returns a pointer to the *Object* structure contained by the *List* pDll.

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> .
------	---	--

## Return Value

The return value from the DllAsObj function is a pointer to the *Object* structure contained by the *List* class.

## Notes

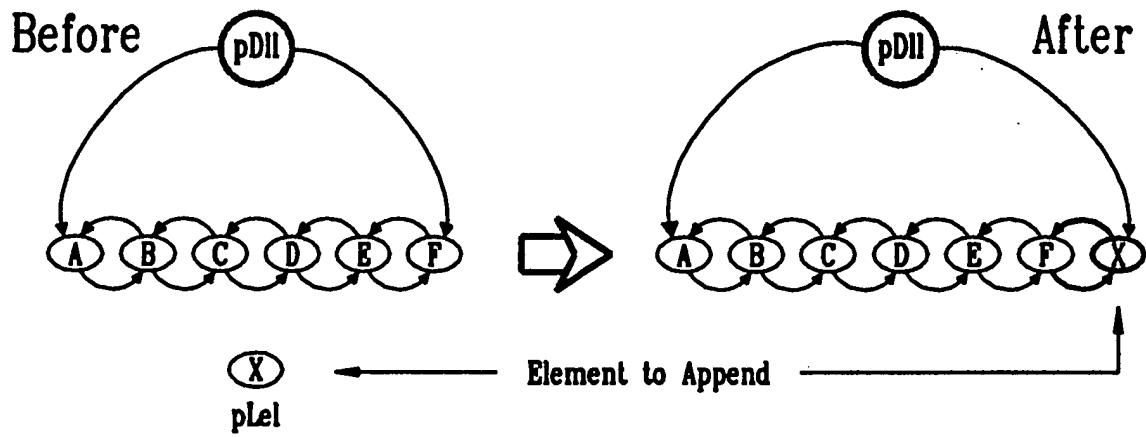
The *Object* pointer can be used to send a message to the client of the *List*.

## Example

Please refer to class test procedure TSTDLL.C,TSTLEL.C for an example of the use of the DllAsObj function.

## DllAppendLast

Diagram



# DllAppendLast

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void      DllAppendLast( pDll, pLel )
PDLL      pDll;
PLEL      pLel;
```

Public Function

A macro is available for this function

## Purpose

The DllAppendLast function links the *ListElement* pLel to the *List* pDll as the last list element.

Parameter	-	Description
-----------	---	-------------

---

pDll	-	Pointer to a structure of type <i>List</i> . The list being linked to.
pLel	-	Pointer to a structure of type <i>ListElement</i> . The list element will be linked as the last list element.

## Return Value

No return value

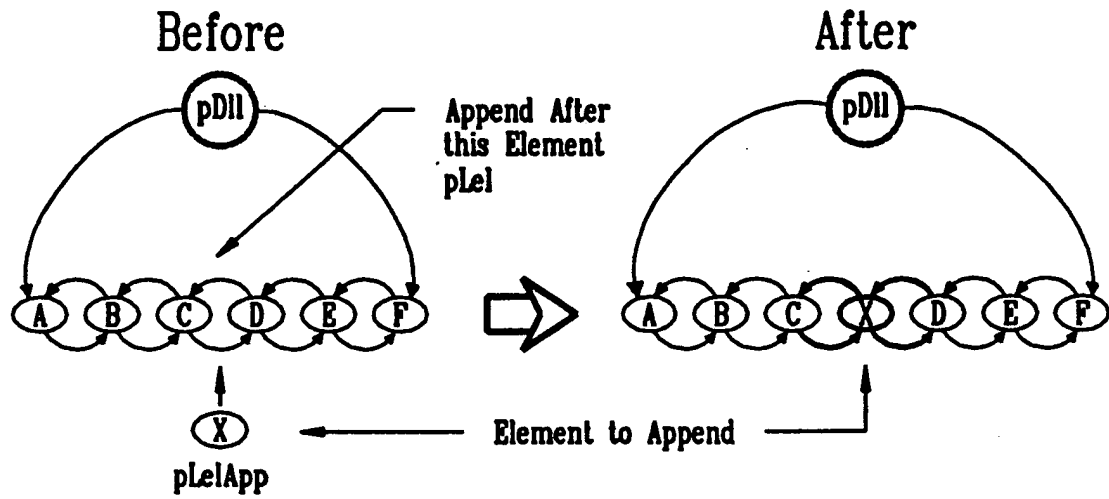
## Notes

pLel cannot already belong to a list.

## See Also

DllAppend, DllInsert, DllInsertFirst

## Diagram





# DllAppend

---

## Summary

```
#include "cobjects.h"
#include "dllmac.h"
```

```
Void      DllAppend( pDll, pLel, pLelApp )
PDLL      pDll;
PLEL      pLel;
PLEL      pLelApp;
```

Public Function

## Purpose

The DllAppend function links the *ListElement* pLelApp, succeeding the *ListElement* pLel, to the *List* pDll. If the *ListElement* pLel is NULL then pLelApp is linked to the *List* pDll as the last list element.

## Parameter - Description

---

pDll	-	Pointer to a structure of type <i>List</i> . The list being linked to.
pLel	-	Pointer to a structure of type <i>ListElement</i> . pLelApp will be linked as the successor of this list element.
pLelApp	-	Pointer to a structure of type <i>ListElement</i> . This is the list element to link to the list.

## Return Value

No return value

## Notes

pLelApp cannot already belong to a list. pLel must already be linked to pDll.

## See Also

DllAppendLast, DllInsert, DllInsertFirst

## Class Reference for *List*

<b>Structure Name:</b>	List
<b>Abbreviation:</b>	DII
<b>Class Type:</b>	Inheritable class

***Class***

---

This page is intentionally left blank

## ClsSendMessageReturnPtr

---

### Notes (cont)

a subobject or superobject of pObj may be the ultimate receiver of the message.

[The message number must be greater than or equal to zero, and less than the number of messages.]

### See Also

ClsSendMessage, ClsSendMessageRetPtr

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsSendMessageReturnPtr function.

# ClsSendMessageReturnPtr

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
Void          ClsSendMessageReturnPtr( pCls, pObj, m, pBlk )
PCLS          pCls;
POBJ          pObj;
MediumInt     m;
PBLK          pBlk;
```

Public Function

## Purpose

The ClsSendMessageRetPtr function sends the message indicated by the selector index *m* to the *Object* *pObj* which is an object of *Class* *pCls*. Parameters are sent to the method through the *Block* *pBlk*. The return value is the return value from the method called.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
pObj	-	Pointer to a structure of type <i>Object</i> . The object receiving the message.
m	-	The selector index.
pBlk	-	Pointer to a structure of type <i>Block</i> . Optional parameters to pass to the method.

range: *m* must be in the range [0:N-1] where *N* is the number of messages *pCls* responds to.

## Return Value

No return value

## Notes

If *pBlk* has a method set it is ignored and the is set to the method indicated by the message index *m*.

*pObj* is not necessarily the precise object which receives the message. *pObj* is offset by an amount determined by the message first. Therefore,

## ClsSendMessageReturnInt

---

### Notes (cont)

pObj is offset by an amount determined by the message first. Therefore, a subobject or superobject of pObj may be the ultimate receiver of the message.

[The message number must be greater than or equal to zero, and less than the number of messages.]

### See Also

ClsSendMessage, ClsSendMessageRetPtr

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsSendMessageReturnInt function.

# ClsSendMessageReturnInt

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsSendMessageReturnInt( pCls, pObj, m, pBlk )
PCLS         pCls;
POBJ         pObj;
MediumInt    m;
PBLK         pBlk;
```

Public Function

## Purpose

The ClsSendMessageRetInt function sends the message indicated by the selector index *m* to the *Object* *pObj* which is an object of *Class* *pCls*. Parameters are sent to the method through the *Block* *pBlk*. The return value is the return value from the method called.

## Parameter - Description

---

<i>pCls</i>	-	Pointer to a structure of type <i>Class</i> .
<i>pObj</i>	-	Pointer to a structure of type <i>Object</i> . The object receiving the message.
<i>m</i>	-	The selector index.
<i>pBlk</i>	-	Pointer to a structure of type <i>Block</i> . Optional parameters to pass to the method.

range: *m* must be in the range [0:N-1] where *N* is the number of messages *pCls* responds to.

## Return Value

The return value from the ClsSendMessageRetInt function is the return value from the method which is called.

## Notes

If *pBlk* has a method set it is ignored and the is set to the method indicated by the message index *m*.

*pObj* is not necessarily the precise object which receives the message.

## ClsSendMessage

---

### Notes (cont)

[The message number must be greater than or equal to zero, and less than the number of messages.]

### See Also

ClsSendMessageRetInt, ClsSendMessageRetPtr

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsSendMessage function.



# ClsSendMessage

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
Void          ClsSendMessage( pCls, pObj, m, pBlk )
PCLS          pCls;
POBJ          pObj;
MediumInt     m;
PBLK          pBlk;
```

Public Function

## Purpose

The ClsSendMessage function sends the message indicated by the selector index *m* to the *Object* *pObj* which is an object of *Class* *pCls*. Parameters are sent to the method through the *Block* *pBlk*.

## Parameter - Description

---

<i>pCls</i>	-	Pointer to a structure of type <i>Class</i> .
<i>pObj</i>	-	Pointer to a structure of type <i>Object</i> . The object receiving the message.
<i>m</i>	-	The selector index.
<i>pBlk</i>	-	Pointer to a structure of type <i>Block</i> . Optional parameters to pass to the method.

range: *m* must be in the range [0:N-1] where *N* is the number of messages *pCls* responds to.

## Return Value

No return value

## Notes

If *pBlk* has a method set it is ignored and the is set to the method indicated by the message index *m*.

*pObj* is not necessarily the precise object which receives the message. *pObj* is offset by an amount determined by the message first. Therefore, a subobject or superobject of *pObj* may be the ultimate receiver of the message.

## ClsSendDestroy

---

### Summary

```
#include "cobjects.h"  
#include "clsmac.h"
```

```
Void          ClsSendDestroy( pCls )  
PCLS          pCls;
```

Public Function

### Purpose

The ClsSendMessage destroys the *Class* pCls with the proper *Class* destructor function.

### Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

### Return Value

No return value

### See Also

ClsDestroy, ClsDeInit, MclSendCreateClass, MclSendDestroyClass

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsSendDestroy function.

# ClsPrint

---

## Summary

```
#include "objects.h"
#include "clsmac.h"
```

```
Void          ClsPrint( pCls, pCio, item, level, name )
PCLS          pCls;
PCIO          pCio;
MediumInt     item;
MediumInt     level;
PSTR          name;
```

Public Function

## Purpose

The ClsPrint function prints the contents of the *Class* pCls on the ConsoleInputOutput device pCio. The item parameter is the array index of this instance or -1 if it is not an array element, level is a number indicating the level of indentation, and name is a *String* pointer which is the name of this instance.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> . The <i>Class</i> being searched.
pCio	-	Pointer to a structure of type ConsoleInputOutput.
item	-	The array element number of this instance (or -1).
level	-	The level of indentation to print this object with.
name	-	The name of this instance.

## Return Value

No return value

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsPrint function.

## Summary

```
#include "cobjects.h"  
#include "clsmac.h"
```

```
Bool      ClsIsRoot( pCls )  
PCLS      pCls;
```

Public Function

## Purpose

The ClsIsRoot function returns True if the *Class* pCls is the root subclass, otherwise False is returned.

## Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

## Return Value

The return value from the ClsIsRoot function is True if pCls is the root subclass otherwise False is returned.

## Notes

A root subclass is a class which has no subclasses.

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsIsRoot function.

# ClsInit

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
Void          ClsInit( pCls, pMcl )
PCLS          pCls;
PMCL          pMcl;
```

Public Function

## Purpose

The ClsInit function initializes the *Class* pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

No return value

## Notes

If you create a *Class* with the MclSendCreateClass function, you do not need to call ClsInit as it is called for you. However, if pCls points to statically allocated memory, you must call ClsInit before attempting to use any other *Class* functions. When you are done with pCls, you should deinitialize it with ClsDeInit.

## See Also

ClsDeInit, ClsDestroy, MclSendCreateClass, MclSendDestroyClass

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsInit function.

# ClsGetSuperClassIndex

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsGetSuperClassIndex( pCls )
PCLS         pCls;
```

Public Function

## Purpose

The ClsGetSuperClassIndex function returns the index of the superclass pCls. If the *Class* pCls is not a superclass, it returns -1.

## Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

## Return Value

The return value from the ClsGetSuperClassIndex function is the index of the superclass pCls. If pCls is not a superclass -1 is returned.

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetSuperClassIndex function.

# ClsGetSuperClassCount

---

## Summary

```
#include "cobjects.h"  
#include "clsmac.h"
```

```
MediumInt    ClsGetSuperClassCount( pCls )  
PCLS         pCls;
```

Public Function

## Purpose

The ClsGetSuperClassCount function returns the number of superclasses in the *Class* pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

The return value from the ClsGetSuperClassCount function is the number of superclasses in pCls.

## Notes

The range of the return value is [0:30].

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetSuperClassCount function.

# ClsGetSubObjectOffset

---

## Summary

```
#include "cobjects.h"  
#include "clsmac.h"
```

```
MediumInt    ClsGetSubObjectOffset( pCls )  
PCLS         pCls;
```

Public Function

## Purpose

The ClsGetSubObjectOffset function returns the offset in bytes of an object of *Class* pCls from its subclass.

## Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

## Return Value

The return value from the ClsGetSubObjectOffset function is the offset in bytes from an object of pCls to an object of the subclass of pCls.

## Notes

The return value is in the range [-2000:0].

## See Also

ClsGetRootSubObjectOffset

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetSubObjectOffset function.



# ClsGetSubClass

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
PCLS          ClsGetSubClass( pCls )
PCLS          pCls;
```

Public Function

## Purpose

The ClsGetSubClass function returns the subclass of the *Class* pCls or NULL if the *Class* pCls has no subclass.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

The return value from the ClsGetSubClass function is a pointer to a structure of type *Class* which is the subclass of pCls. If pCls has no subclass, NULL is returned.

## Notes

If NULL is returned, then pCls is the root *Class*.

## See Also

ClsGetRootSubClass, ClsGetNthSuperClass

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetSubClass function.

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt   ClsGetSize( pCls )
PCLS        pCls;
```

Public Function

## Purpose

The ClsGetSize function returns the size of the *Class* pCls in bytes.

## Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

## Return Value

The return value from the ClsGetSize function is the size of the *Class* pCls in bytes.

## Notes

Typically, the size of a class is sizeof( *Class* ), however you can create classes which have larger sizes (for holding class specific data). See the section on *MetaClass* for further details.

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetSize function.

# ClsGetRootSubObjectSize

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsGetRootSubObjectSize( pCls )
PCLS         pCls;
```

Public Function

## Purpose

The ClsGetRootSubObjectSize function returns the size in bytes of a root object described by the *Class* pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

The return value from the ClsGetRootSubObjectSize function is the size of a root object of pCls in bytes.

## Notes

The root *Object* is an instance of the root *Class* which is the subclass of pCls which has no subclass.

## See Also

ClsGetObjectSize, ClsGetRootSubClass, ClsGetRootSubObjectOffset

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetRootSubObjectSize function.

# ClsGetRootSubObjectOffset

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsGetRootSubObjectOffset( pCls )
PCLS         pCls;
```

Public Function

## Purpose

The ClsGetRootSubObjectOffset function returns the offset in bytes of the root object described by the *Class* pCls.

Parameter	-	Description
-----------	---	-------------

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

The return value from the ClsGetRootSubObjectOffset function is the offset of a root object of *Class* pCls in bytes.

## Notes

The root *Object* is an instance of the root *Class* which is the subclass of pCls which has no subclass.

The return value is in the range [-2000:0].

## See Also

ClsGetObjectSize, ClsGetRootSubClass, ClsGetRootSubObjectSize

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetRootSubObjectOffset function.

# ClsGetRootSubClass

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
PCLS          ClsGetRootSubClass( pCls )
PCLS          pCls;
```

Public Function

## Purpose

The ClsGetRootSubClass function returns a pointer to a structure of type *Class* which is the outermost subclass of the *Class* pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

The return value from the ClsGetRootSubClass function is a pointer to a structure of type *Class* which is the outermost subclass of the *Class* pCls.

## Notes

By definition, the returned *Class* cannot have a subclass.

The root subclass of the returned value is itself.

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetRootSubClass function.

## ClsGetOffsetOfNthSuper

---

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetOffsetOfNthSuper function.

# ClsGetOffsetOfNthSuper

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsGetOffsetOfNthSuper( pCls, n )
PCLS         pCls;
MediumInt    n;
```

Public Function

## Purpose

The ClsGetOffsetOfNthSuper function returns the offset of the object (relative to an object of the *Class* pCls) which is the Nth superclass of pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
n	-	The index of the superclass.

range: n must be in the range [0:N-1] where N is the number of superclasses of pCls.

## Return Value

The return value from the ClsGetOffsetOfNthSuper function is the offset from an *Object* of *Class* pCls needed to get to the Nth superclass.

## Notes

The return value will be in the range [0:2000].

[The super class index must be greater than or equal to zero, and less than the number of superclasses.]

## See Also

ClsGetNthSuperClass

## ClsGetOffsetForMsg

---

### See Also

ClsGetMethodAndOffset, ClsGetNthMsg

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetOffsetForMsg function.



# ClsGetOffsetForMsg

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsGetOffsetForMsg( pCls, m )
PCLS         pCls;
MediumInt     m;
```

Public Function

## Purpose

The ClsGetOffsetForMsg function returns the offset of the object (relative to an object of the *Class* pCls) which ultimately responds to the message index m.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
m	-	The index of the message selector.

range: m must be in the range [0:N-1] where N is the number of messages to which pCls can respond to.

## Return Value

The return value from the ClsGetOffsetForMsg function is the offset of the object which ultimately handles the message m.

## Notes

The return value will be in the range [-2000:+2000]. A negative value indicates the message is being overridden by a subclass. A positive value means a superclass function is being inherited. A value of 0 means pCls is defining/overriding the method.

[The message number must be greater than or equal to zero, and less than the number of messages.]

# ClsGetObjectSize

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsGetObjectSize( pCls )
PCLS         pCls;
```

Public Function

## Purpose

The ClsGetObjectSize function returns the size in bytes of an (super) object described by the (super) *Class* pCls.

## Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

## Return Value

The return value from the ClsGetObjectSize function is the size of an object of this class in bytes.

## Notes

To get the size of the entire (root) object, not just the superobject, use the function ClsGetRootObjectSize.

## See Also

ClsGetRootObjectSize

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetObjectSize function.

# ClsGetNthSuperClass

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
PCLS          ClsGetNthSuperClass( pCls, n )
PCLS          pCls;
MediumInt     n;
```

Public Function

## Purpose

The ClsGetNthSuperClass function returns a pointer to a structure of type *Class* which is the *n*th superclass of the *Class* pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
n	-	The index of the superclass.

range: n must be in the range [0:N-1] where N is the number of superclasses of pCls.

## Return Value

The return value from the ClsGetNthSuperClass function is a pointer to a structure of type *Class* which is the Nth superclass of pCls.

## Notes

[The super class index must be greater than or equal to zero, and less than the number of superclasses.]

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetNthSuperClass function.

# ClsGetNthMsg

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
PMSG      ClsGetNthMsg( pCls, n )
PCLS      pCls;
MediumInt n;
```

Public Function

## Purpose

The ClsGetNthMsg function returns a pointer to a structure of type *Message* given the index *m* for the *Class* *pCls*.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
m	-	The index of the message

range: *m* must be in the range [0:N-1] where *N* is the number of messages *pCls* responds to.

## Return Value

The return value from the ClsGetNthMsg function is a pointer to a structure of type *Message* which is the *Nth* message of *pCls*.

## Notes

[The message number must be greater than or equal to zero, and less than the number of messages.]

## See Also

ClsGetMethodAndOffset

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetNthMsg function.

# ClsGetName

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
PSTR      ClsGetName( pCls )
PCLS      pCls;
```

Public Function

## Purpose

The ClsGetName function returns a pointer to a *String* containing the name of the *Class* pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

The return value from the ClsGetName function is a pointer to a *String* containing the name of the *Class* pCls.

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetName function.

## ClsGetMethodAndOffset

---

### Notes (cont)

ppMth will point to a function after this call.

[The message number must be greater than or equal to zero, and less than the number of messages.]

### See Also

ClsGetNthMessage

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetMethodAndOffset function.

# ClsGetMethodAndOffset

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
Void          ClsGetMethodAndOffset( pCls, m, ppMth, pOffset )
PCLS          pCls;
MediumInt     m;
PMTH          *ppMth;
MediumInt     *pOffset;
```

Public Function

## Purpose

The ClsGetMethodAndOffset function returns a pointer to the Method and the offset of the object (relative to this class) which owns the method. The Method pointer is returned in ppMth and the offset is returned in pOffset. The method and offset are determined by the message index m within the Class pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
m	-	The index of the message selector.
ppMth	-	Pointer to a pointer to a Method function. The method is returned here.
pOffset	-	Pointer to a MediumInt. The offset is returned here.

range: \*pOffset will be in the range [-2000:+2000]. A negative value indicates the message is being overridden by a subclass. A positive value means a superclass function is being inherited. A value of 0 means pCls is defining/overriding the method.

## Return Value

No return value

## Notes

m must be in the range [0:N-1] where N is the number of messages to which pCls can respond to.

# ClsGetMessageCount

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
MediumInt    ClsGetMessageCount( pCls )
PCLS         pCls;
```

Public Function

## Purpose

The ClsGetMessageCount function returns the number of messages the *Class* pCls can respond to.

## Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

Range: The range of the return value is [0:SHRT\_MAX] where SHRT\_MAX is defined in the include file <limits.h>.

## Return Value

The return value from the function ClsGetMessageCount function is the number of messages pCls responds to.

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsGetMessageCount function.



# ClsFindSuperClass

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
PCLS      ClsFindSuperClass( pCls, pStr )
PCLS      pCls;
PSTR      pStr;
```

Public Function

## Purpose

The ClsFindSuperClass function finds searches the *Class* pCls for the superclass identified by the string pStr and returns a pointer to the superclass if found and NULL if not found.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
pStr	-	Pointer to a <i>String</i> containing the superclass name.

## Return Value

The return value from the function ClsFindSuperClass is a pointer to a structure of type *Class* which is the superclass identified by pStr. NULL is returned if it is not found.

## Notes

The search is case-sensitive.

Superclass names should generally be named the same as the structure that they describe.

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsFindSuperClass function.

## ClsFindSelectorIndex

---

### See Also

ClsFindSelectorIndex, ClsSendMessage

### Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsFindSelectorIndex function.

# ClsFindSelectorIndex

---

## Summary

```
#include "cobjects.h"  
#include "clsmac.h"
```

```
MediumInt    ClsFindSelectorIndex( pCls, pStr )  
PCLS         pCls;  
PSTR         pStr;
```

Public Function

## Purpose

The ClsFindSelectorIndex function searches the *Class* pCls for the message selector string pStr and returns the index of the selector if found and -1 if not found.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
pStr	-	Pointer to a <i>String</i> containing the selector name.

range: The return value will be in the range [0:N-1] where N is the number of messages pCls responds to.

## Return Value

The return value from the ClsFindSelectorIndex function is the index of the message if found and -1 if not found.

## Notes

The search is case-sensitive.

Selector strings should generally be by mixed case without spaces and with the first character lowercase.

The return value can subsequently be used in the ClsSendMessage function to send a message to an object.

# ClsFindMsg

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
PMSG      ClsFindMsg( pCls, pStr )
PCLS      pCls;
PSTR      pStr;
```

Public Function

## Purpose

The ClsFindMsg function finds searches the *Class* pCls for the message selector pStr and returns a pointer to the *Message* if found or NULL if not found.

Parameter	-	Description
-----------	---	-------------

---

pCls	-	Pointer to a structure of type <i>Class</i> .
pStr	-	Pointer to a <i>String</i> containing the selector name.

## Return Value

The return value from the ClsFindMsg function is a pointer to a structure of type *Message* which contains the message selector pStr. NULL is returned if it is not found.

## Notes

The search is case-sensitive.

Selector strings should generally be by mixed case without spaces and with the first character lowercase.

## See Also

ClsFindSelectorIndex

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsFindMsg function.

# **ClsDestroyObject**

---

## **Example**

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsDestroyObject function.

# ClsDestroyObject

---

## Summary

```
#include "cobjects.h"  
#include "clsmac.h"
```

```
Void          ClsDestroyObject( pCls, pObj )  
PCLS          pCls;  
POBJ          pObj;
```

Public Function

## Purpose

The ClsDestroyObject function deallocates the memory associated with the *Object* pObj whose type is specified by the *Class* pCls.

## Parameter - Description

---

pCls	-	Pointer to a structure of type <i>Class</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .

## Return Value

No return value

## Notes

pObj must have been deinitialized prior to calling this function.

pCls must be a root class, i.e. it has no subclasses.

pObj must have been allocated by ClsCreateObject.

[The class must not have any sub-classes.]

## See Also

ClsCreateObject

# ClsDestroy

---

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
Void          ClsDestroy( pCls )
PCLS          pCls;
```

Public Function

## Purpose

The ClsDestroy function deinitializes the *Class* pCls and deallocates the memory used by it. The *Class* pCls should not be referenced after this function is called.

Parameter	-	Description
-----------	---	-------------

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

No return value

## Notes

Any superclasses owned by pCls are destroyed as well.

Any messages owned by pCls are destroyed as well.

pCls should point to a *Class* which was created by MclSendCreateClass. If pCls points to statically allocated memory then the caller should use ClsDeInit instead of this function.

## See Also

ClsDeInit, ClsInit, MclSendCreateClass, MclSendDestroyClass

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsDestroy function.

## Summary

```
#include "cobjects.h"
#include "clsmac.h"
```

```
Void          ClsDeInit( pCls )
PCLS          pCls;
```

Public Function

## Purpose

The ClsDeInit function deinitializes the *Class* pCls. The ClsDeInit function should be the last function called when done using pCls and just prior to freeing its memory.

Parameter	-	Description
-----------	---	-------------

---

pCls	-	Pointer to a structure of type <i>Class</i> .
------	---	---

## Return Value

No return value

## Notes

If you use MclSendCreateClass to create a *Class* and ClsDestroy to get rid of it you will never need to use this function. It is of value only if pCls points to a statically allocated structure which was initialized with ClsInit.

## See Also

ClsDestroy, ClsInit, MclSendCreateClass, MclSendDestroyClass

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsDeInit function.



# ClsCreateObject

---

## Summary

```
#include "cobjects.h"  
#include "clsmac.h"
```

```
POBJ      ClsCreateObject( pCls )  
PCLS      pCls;
```

Public Function

## Purpose

The ClsCreateObject function creates a new *Object* of the type specified by the *Class* pCls.

## Parameter - Description

---

pCls - Pointer to a structure of type *Class*.

## Return Value

The return value from the ClsCreateObject function is a pointer to a structure of type *Object* and is the new object instance.

## Notes

The pointer returned by ClsCreateObject actually is a pointer to two types of structure. It is a pointer to the type which is described by the *Class* pCls and it is a pointer to type *Object*. The reason for this is that all structures which either inherit or are inherited from are defined as having their first structure member be a structure of type *Object*. In this way, the two structures are superimposed upon one another.

[The class must not have any sub-classes.]

## Example

Please refer to class test procedure TSTCLS.C for an example of the use of the ClsCreateObject function.

## Class Reference for *Class*

<b>Structure Name:</b>	Class
<b>Abbreviation:</b>	Cls
<b>Class Type:</b>	Primitive Class

# BlkSetMethod

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void      BlkSetMethod( pBlk, pMth )
PBLK      pBlk;
PMTH      pMth;
```

Public Function

## Purpose

The BlkSetMethod function saves the pointer to Method pMth as the function to call when executing the *Block* pBlk.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pMth	-	Pointer to a Method.

## Return Value

No return value

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkSetMethod function.

## BlkPushMediumInt

---

### Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void          BlkPushMediumInt( pBlk, p )
PBLK          pBlk;
MediumInt     p;
```

Public Function

### Purpose

The BlkPushMediumInt function pushes the value **p** onto the parameter *Block* pBlk.

Parameter	-	Description
-----------	---	-------------

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
p	-	The integer parameter being pushed.

### Return Value

No return value

### Notes

[The total number of parameters must not be larger than the parameter array.]

### Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkPushMediumInt function.

# BlkPushLargeInt

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void          BlkPushLargeInt( pBlk, p )
PBLK          pBlk;
LargeInt      p;
```

Public Function

## Purpose

The BlkPushLargeInt function pushes the value **p** onto the parameter *Block* pBlk.

Parameter	-	Description
-----------	---	-------------

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
p	-	The integer parameter being pushed.

## Return Value

No return value

## Notes

[The total number of parameters must not be larger than the parameter array.]

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkPushLargeInt function.

## BlkPushFuncPtr

---

### Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void      BlkPushFuncPtr( pBlk, p )
PBLK      pBlk;
PMTH      p;
```

Public Function

### Purpose

The BlkPushFuncPtr function pushes the function pointer Method **p** onto the *Block* pBlk.

### Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
p	-	Pointer to Method. The pointer parameter being pushed.

### Return Value

No return value

### Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkPushFuncPtr function.

# BlkPushDataPtr

---

## Summary

```
#include "cobjects.h"
```

```
#include "blkmac.h"
```

```
Void          BlkPushDataPtr( pBlk, p )  
PBLK          pBlk;  
PMEM          p;
```

Public Function

## Purpose

The BlkPushDataPtr function pushes the data pointer *Memory* **p** onto the *Block* pBlk.

Parameter	-	Description
-----------	---	-------------

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
p	-	Pointer to Mem. The pointer parameter being pushed.

## Return Value

No return value

## Notes

[The total number of parameters must not be larger than the parameter array.]

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkPushDataPtr function.

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void          BlkPrint( pBlk, pCio, item, level, name )
PBLK          pBlk;
PCIO          pCio;
MediumInt     item;
MediumInt     level;
PSTR          name;
```

Public Function

## Purpose

The BlkPrint function prints the contents of the *Block* pBlk on the ConsoleInputOutput device pCio. The item parameter is the array index of this instance or -1 if it is not an array element, level is a number indicating the level of indentation, and name is a *String* pointer which is the name of this instance.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pCio	-	Pointer to a structure of type ConsoleInputOutput.
item	-	The array element number of this instance (or -1).
level	-	The level of indentation to print this object with.
name	-	The name of this instance.

## Return Value

No return value

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkPrint function.



# BlkInit

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void      BlkInit( pBlk, pMth )
PBLK      pBlk;
PMTH      pMth;
```

Public Function

## Purpose

The BlkInit function initializes the *Block* pBlk and sets its method to the Method pointer pMth. This should be the first function called after pBlk has been allocated.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pMth	-	Pointer to a Method.

## Return Value

No return value

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkInit function.

# BlkHasMethod

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Bool      BlkHasMethod( pBlk )
PBLK      pBlk;
```

Public Function

## Purpose

The BlkHasMethod function returns True if the *Block* pBlk has a non-NULL Method pointer associated with it, otherwise False is returned.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
------	---	---

## Return Value

The return value from the BlkHasMethod function is True if pBlk has a non-NULL value and False if it is NULL.

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkHasMethod function.

# BlkExecuteRetInt

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
MediumInt    BlkExecuteRetInt( pBlk, pObj )
PBLK         pBlk;
POBJ         pObj;
```

Public Function

## Purpose

The BlkExecuteRetInt function executes the method in the *Block* pBlk and passes parameters to it. The first parameter passed will be the *Object* pObj. The return value is a MediumInt returned from the method.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .

## Return Value

The return value of BlkExecuteRetInt is a MediumInt returned from the method executed.

## Notes

[pBlk must contain a non NULL method pointer.]

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkExecuteRetInt function.

# BlkExecuteRetFuncPtr

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
PMTH      BlkExecuteRetFuncPtr( pBlk, pObj )
PBLK      pBlk;
POBJ      pObj;
```

Public Function

## Purpose

The BlkExecuteRetFuncPtr function executes the method in the *Block* pBlk and passes parameters to it. The first parameter passed will be the *Object* pObj. The return value is a Method pointer returned from the method.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .

## Return Value

The return value from the BlkExecuteRetFuncPtr function is a Method pointer returned from the method executed.

## Notes

[pBlk must contain a non NULL method pointer.]

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkExecuteRetFuncPtr function.

# BlkExecuteRetDataPtr

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void      BlkExecuteRetDataPtr( pBlk, pObj )
PBLK      pBlk;
POBJ      pObj;
```

Public Function

## Purpose

The BlkExecuteRetDataPtr function executes the method in the *Block* pBlk and passes any parameters to it. The first parameter passed will be the *Object* pObj. The return value is a data pointer (Void \*) returned from the method.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .

## Return Value

No return value

## Notes

The return value from the BlkExecuteRetDataPtr function is a data pointer (Void \*) returned from the method executed.

[pBlk must contain a non NULL method pointer.]

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkExecuteRetDataPtr function.

# BlkExecuteRetBool

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Bool      BlkExecuteRetBool( pBlk, pObj )
PBLK      pBlk;
POBJ      pObj;
```

Public Function

## Purpose

The BlkExecuteRetBool function executes the method in the *Block* pBlk and passes any parameters to it. The first parameter passed will be the *Object* pObj. The return value is a Bool returned from the method.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .

## Return Value

The return value from the BlkExecuteRetBool function is a boolean (True/False) value returned from the method executed.

## Notes

[pBlk must contain a non NULL method pointer.]

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkExecuteRetBool function.

# BlkExecute

---

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void          BlkExecute( pBlk, pObj )
PBLK          pBlk;
POBJ          pObj;
```

Public Function

## Purpose

The BlkExecute function executes the method in the *Block* pBlk and passes any parameters to it. The first parameter passed will be the *Object* pObj.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
pObj	-	Pointer to a structure of type <i>Object</i> .

## Return Value

No return value

## Notes

[pBlk must contain a non NULL method pointer.]

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkExecute function.

## Summary

```
#include "cobjects.h"
#include "blkmac.h"
```

```
Void      BlkDeInit( pBlk )
PBLK      pBlk;
```

Public Function

## Purpose

The BlkDeInit function deinitializes the *Block* pBlk. It should be the last call referencing pBlk before deallocating it.

Parameter	-	Description
-----------	---	-------------

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
------	---	---

## Return Value

No return value

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkDeInit function.



# BlkClear

---

## Summary

```
#include "cobjects.h"  
#include "blkmac.h"
```

```
Void      BlkClear( pBlk )  
PBLK      pBlk;
```

Public Function

## Purpose

The BlkClear function clears the parameters of the *Block* pBlk.

## Parameter - Description

---

pBlk	-	Pointer to a structure of type <i>Block</i> .
------	---	---

## Return Value

No return value

## Example

Please refer to class test procedure TSTBLK.C for an example of the use of the BlkClear function.

## Class Reference for *Block*

**Structure Name:** Block  
**Abbreviation:** Blk  
**Class Type:** Primitive Class

## C+O Datatypes

---

This page is intentionally left blank

## C+O Datatypes

Name	Size	Definition	Usage
Reg2	N/A	register	Prioritized register allocation
Reg3	N/A	register	Prioritized register allocation
Reg4	N/A	register	Prioritized register allocation
Reg5	N/A	register	Prioritized register allocation
SmallInt	1	char	Integers in range [-127:+127]
Str	1	char	Text data
String	1	char	Text data
Task	24/32	struct <i>Task</i>	<i>Task</i> class
Tre	16/32	struct <i>Tree</i>	<i>Tree</i> class
Tree	16/32	struct <i>Tree</i>	<i>Tree</i> class
True	N/A	((Bool)1)	Boolean True
Tsk	24/32	struct <i>Task</i>	<i>Task</i> class
ULargeInt	4	unsigned long	Integers in the range [0:0xFFFFFFFF]
UMediumInt	2	unsigned int	Integers in the range [0:0xFFFF]
USmallInt	1	unsigned char	Integers in range [0:0xFF]
Vertex	22/44	struct <i>Vertex</i>	<i>Vertex</i> class
Void	N/A	void	Function declarations
Volatile	N/A	volatile	(default) Identifies function parameters which are modified
Vtx	22/44	struct <i>Vertex</i>	<i>Vertex</i> class

# C+O Datatypes

Name	Size	Definition	Usage
Mem	1	char	<i>Memory</i> class
<i>Memory</i>	1	char	<i>Memory</i> class
<i>Message</i>	12/22	struct <i>Message</i>	<i>Message</i> class
<i>MetaClass</i>	20/20	struct <i>MetaClass</i>	<i>MetaClass</i> class
<i>MetaMessage</i>	8/14	struct <i>MetaMessage</i>	<i>MetaMessage</i> class
<i>MetaSuperClass</i>	12/22	struct <i>MetaSuperClass</i>	<i>MetaSuperClass</i> class
Method	2/4	Void (*)( POBJ, ... )	Function pointer (see <i>Block</i> )
MethodRetBool	2/4	Bool (*)( POBJ, ... )	Function (returning Bool) pointer (see <i>Block</i> )
MethodRetInt	2/4	MediumInt (*)( POBJ, ... )	Function (returning MediumInt) pointer (see <i>Block</i> )
MethodRetDataPtr	2/4	PMEM (*)( POBJ, ... )	Function (returning data pointer) pointer (see <i>Block</i> )
MethodRetFuncPtr	2/4	PMTH (*)( POBJ, ... )	Function (returning function pointer) pointer (see <i>Block</i> )
MethodRetPtr	2/4	Void * (*)( POBJ, ... )	Function (returning pointer) pointer (see <i>Block</i> )
Mms	8/14	struct <i>MetaMessage</i>	<i>MetaMessage</i> class
MODULE_NAME	N/A	__FILE__	Name of file being compiled
Msc	12/22	struct <i>MetaSuperClass</i>	<i>MetaSuperClass</i> class
Msg	12/22	struct <i>Message</i>	<i>Message</i> class
NULL	2/4	0	Used to assign or return null pointers
Obj	2/4	struct <i>Object</i>	<i>Object</i> class
<i>Object</i>	2/4	struct <i>Object</i>	<i>Object</i> class
PBLK	2/4	struct <i>Block</i> *	<i>Block</i> pointer - does not require structure definition
PCIO	2/4	struct ConsoleInputOutput *	ConsoleInputOutput pointer - does not require structure definition
PCLS	2/4	struct <i>Class</i> *	<i>Class</i> pointer - does not require structure definition
PDLL	2/4	struct <i>List</i> *	<i>List</i> pointer - does not require structure definition
PDPA	2/4	struct <i>DynamicArray</i> *	<i>DynamicArray</i> pointer - does not require structure definition
PEDG	2/4	struct <i>Edge</i> *	<i>Edge</i> pointer - does not require structure definition
PGRF	2/4	struct <i>Graph</i> *	<i>Graph</i> pointer - does not require structure definition
PJUL	2/4	struct <i>JulianTime</i> *	<i>JulianTime</i> pointer - does not require structure definition
PLEL	2/4	struct <i>ListElement</i> *	<i>ListElement</i> pointer - does not require structure definition
PMCL	2/4	struct <i>MetaClass</i> *	<i>MetaClass</i> pointer - does not require structure definition
PMEM	2/4	Char *	Mem pointer - does not require structure definition
PMMS	2/4	struct <i>MetaMessage</i> *	<i>MetaMessage</i> pointer - does not require structure definition
PMSC	2/4	struct <i>MetaSuperClass</i> *	<i>MetaSuperClass</i> pointer - does not require structure definition
PMSG	2/4	struct <i>Message</i> *	<i>Message</i> pointer - does not require structure definition
PMTH	2/4	Void (*)( POBJ, ... )	Function pointer (see <i>Block</i> )
POBJ	2/4	struct <i>Object</i> *	<i>Object</i> pointer - does not require structure definition. Also synonymous with a class pointer of unknown type
PSTR	2/4	char *	Null terminated text strings
PTRE	2/4	struct <i>Tree</i> *	<i>Tree</i> pointer - does not require structure definition
PTSK	2/4	struct <i>Task</i> *	<i>Task</i> pointer - does not require structure definition
PVTX	2/4	struct <i>Vertex</i> *	<i>Vertex</i> pointer - does not require structure definition
Real	8	double	Floating point (no range specified)
Reg1	N/A	register	Prioritized register allocation

# C+O Datatypes

Name	Size	Definition	Usage
Blk	20/38	struct <i>Block</i>	<i>Block</i> class
<i>Block</i>	20/38	struct <i>Block</i>	<i>Block</i> class
Bool	1	char	Boolean values: True or False
Call	N/A	(Void)	Overrides function returning value
Char	1	char	Text data
Chr	1	char	Text data
<i>Class</i>	16/26	struct <i>Class</i>	<i>Class</i> class
ClientPtr	2/4	Void *	Data pointer to unknown type
Cls	16/26	struct <i>Class</i>	<i>Class</i> class
Const	N/A	const	Identifies function parameters which are not modified
DateFormat	2	enum DateFormat	Describe a display format for dates (see <i>JulianTime</i> and <i>String</i> )
Dll	6/12	struct <i>List</i>	<i>List</i> class
Dpa	8/10	struct <i>DynamicArray</i>	<i>DynamicArray</i> class
<i>DynamicArray</i>	8/10	struct <i>DynamicArray</i>	<i>DynamicArray</i> class
Edg	26/52	struct <i>Edge</i>	<i>Edge</i> class
<i>Edge</i>	26/52	struct <i>Edge</i>	<i>Edge</i> class
ExceptionType	2	enum ExceptionType	Type of exception generated (see <i>Task</i> )
ExcFilter	2/4	Bool (*)(PTSK)	Function pointer to exception filters (see <i>Task</i> )
Ext	2	enum ExceptionType	Type of exception generated
False	N/A	((Bool)0)	Boolean False
Flags8	1	unsigned char	Bit flags (8)
Flags16	2	unsigned short	Bit flags (16)
Flags32	4	unsigned char	Bit flags (32)
GenericPtr	2/4	Void *	Data pointer to unknown type
<i>Graph</i>	30/48	struct <i>Graph</i>	<i>Graph</i> class
Grt	30/48	struct <i>Graph</i>	<i>Graph</i> class
IntAddress	2/4	int / long	Integer capable of holding a data address
Jul	4	struct <i>JulianTime</i>	<i>JulianTime</i> class
<i>JulianTime</i>	4	struct <i>JulianTime</i>	<i>JulianTime</i> class
LARGE_DATA_PTRS	N/A	N/A	Defined if 4 byte data pointers are the default
LARGE_FUNC_PTRS	N/A	N/A	Defined if 4 byte function pointers are the default
LargeInt	4	long	Integers in the range [-2147483647:2147483647]
Lel	8/16	struct <i>ListElement</i>	<i>ListElement</i> class
LINE_NO	N/A	__LINE__	Line number in file being compiled
<i>List</i>	6/12	struct <i>List</i>	<i>List</i> class
<i>ListElement</i>	8/16	struct <i>ListElement</i>	<i>ListElement</i> class
Mcl	20/30	struct <i>MetaClass</i>	<i>MetaClass</i> class
MediumInt	2	int	Integers in range [-32767:32767]

## Quick Reference Guide to Classes

---

This page is intentionally left blank

## Quick Reference Guide to Classes

---

Function Name	Description	Page	Scope	Macro
VtxGetFirstIn	Return first incoming edge	Vtx - 22	Public	Y
VtxGetFirstOut	Return first outgoing edge	Vtx - 23	Public	Y
VtxGetGrf	Return graph	Vtx - 24	Friend	Y
VtxInGrf	Is vertex in graph	Vtx - 26	Public	Y
VtxInit	Initialize the Vertex object	Vtx - 25	Public	N
VtxSendDestroy	Send message for vertex destruction	Vtx - 27	Public	N
VtxStackSetup	Set up values for topsort	Vtx - 28	Friend	Y
VtxVisitEdge	Visit function: each edge	Vtx - 29	Friend	N
VtxVisitEdgeClient	Visit function: each edge	Vtx - 31	Public	N
VtxVisitInEdge	Visit function: incoming edge	Vtx - 33	Friend	N
VtxVisitInEdgeClient	Visit function: incoming edge	Vtx - 35	Public	N
VtxVisitOutEdge	Visit function: outgoing edge	Vtx - 37	Friend	N
VtxVisitOutEdgeClient	Visit function: outgoing edge	Vtx - 39	Public	N



# Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
TskBasicAssert	Foundation function	N/A	Undoc	N
TskCondition	Raise exception conditionally	Tsk - 2	Public	N
TskDeInit	Deinitialize instance	Tsk - 4	Public	N
TskDefaultInit	Initialize instance with defaults	Tsk - 5	Public	N
TskExit	Exit program with code	Tsk - 6	Public	N
TskExitWithMsg	Exit program and print message	Tsk - 7	Public	N
TskGetArgc	Get main() argument count	Tsk - 8	Public	N
TskGetArgv	Get main() argument vector	Tsk - 9	Public	N
TskGetExceptionCondition	Return exception condition	Tsk - 10	Public	N
TskGetExceptionFileName	Return filename of exception	Tsk - 11	Public	N
TskGetExceptionLineNo	Return exception line number	Tsk - 12	Public	N
TskGetExceptionType	Return exception type	Tsk - 13	Public	N
TskInit	Initialize instance	Tsk - 15	Public	N
TskIsInitialized	Is the Task initialized	N/A	Undoc	N
TskLogCond	Raise exception conditionally	Tsk - 17	Public	N
TskMainLogCond	Raise exception conditionally	Tsk - 19	Public	N
TskMainPreCond	Raise exception conditionally	Tsk - 21	Public	N
TskMainPtrCond	Raise exception conditionally	Tsk - 23	Public	N
TskMainRaiseException	Raise exception unconditionally	Tsk - 25	Public	N
TskNormalExit	Exit task normally	Tsk - 27	Public	N
TskOnException	Establish exception handler	Tsk - 28	Public	N
TskPopExceptionHandler	Pop exception handler	Tsk - 30	Public	N
TskPreCond	Raise exception conditionally	Tsk - 32	Public	N
TskPrintException	Print description of exception	Tsk - 34	Public	N
TskPropagateException	Pass last exception	Tsk - 36	Public	N
TskPtrCond	Raise exception conditionally	Tsk - 38	Public	N
TskPushExh	Push the exception	N/A	Undoc	N
TskRaiseException	Raise exception unconditionally	Tsk - 40	Public	N

VtxAsGrfLel	Return list element in graph	Vtx - 2	Friend	Y
VtxAsInDll	Return vertex as list of incoming edges	Vtx - 3	Friend	N
VtxAsObj	Return edge as object	Vtx - 4	Private	N
VtxAsOutDll	Return vertex as list of outgoing edges	Vtx - 5	Friend	N
VtxClear	Clear vertex	Vtx - 6	Public	N
VtxConnectToGrf	Connect vertex to graph	Vtx - 7	Public	N
VtxCountIn	Count incoming edges	Vtx - 9	Public	Y
VtxCountOut	Count outgoing edges	Vtx - 11	Public	Y
VtxDeInit	Deinitialize the Vertex object	Vtx - 13	Public	N
VtxDestroy	Deinitialize Vertex object and free space	Vtx - 14	Public	N
VtxDisconnectFromGrf	Disconnect vertex from graph	Vtx - 15	Public	Y
VtxFindOutEdg	Visit search function: outgoing edges	Vtx - 17	Private	N
VtxFindOutEdgClient	Visit search function: outgoing edges	Vtx - 19	Public	N
VtxGetClient	Return client of vertex	Vtx - 21	Public	Y

# Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
TreClientVisitLeaves	Visit function: leaves	Tre - 47	Public	N
TreClientVisitParents	Visit function: nearest parents first	Tre - 49	Public	N
TreClientVisitPreOrder	Visit function: pre-order	Tre - 51	Public	N
TreClientVisitRange	Visit function: range	Tre - 53	Public	N
TreClientVisitSuccPreOrder	Visit function: all successors	Tre - 55	Public	N
TreClientVisitSuccessors	Visit function: successors	Tre - 57	Public	N
TreCutChildren	Cut children from tree	Tre - 59	Public	Y
TreCutRange	Cut node(s) from tree	Tre - 61	Public	Y
TreDeInit	Deinitialize Tree object	Tre - 63	Public	N
TreDestroy	Deinitialize Tree object and free space	Tre - 64	Public	N
TreDestroyChildren	Destroy any children of a tre	Tre - 65	Public	N
TreFirstChild	Return first child	Tre - 66	Private	Y
TreHasChildren	Does node have any children	Tre - 68	Public	Y
TreHasSiblings	Does node have any siblings	Tre - 69	Public	Y
TreInit	Initialize tree object	Tre - 70	Public	N
TreIsChild	Does the node have a parent	Tre - 71	Public	Y
TreIsDirectAncestor	Is node a direct ancestor	Tre - 72	Public	N
TreIsRoot	Does the node have no parent	Tre - 73	Public	Y
TreLastChild	Return last child	Tre - 74	Private	Y
TreLastLeaf	Return last leaf	Tre - 76	Private	N
TreNext	Return next node	Tre - 78	Private	Y
TreNextPreOrder	Return next node PreOrderly	Tre - 80	Private	N
TreNextUncle	Return next uncle	Tre - 82	Private	N
TreParent	Return parent node	Tre - 84	Private	Y
TrePasteRangeAfterSibling	Paste range of siblings	Tre - 86	Public	Y
TrePasteRangeBeforeSibling	Paste range of siblings	Tre - 88	Public	Y
TrePasteRangeFirstChild	Paste children	Tre - 90	Public	Y
TrePasteRangeLastChild	Paste children	Tre - 92	Public	Y
TrePrev	Return previous node	Tre - 94	Private	Y
TrePrevPreOrder	Return previous node PreOrderly	Tre - 96	Private	N
TreSendDestroy	Send message for tree destruction	Tre - 98	Public	N
TreVisitBranchInOrder	Visit function: branch in-order	Tre - 99	Private	N
TreVisitChildren	Visit function: children	Tre - 101	Private	N
TreVisitChildrenBwd	Visit function: children	Tre - 103	Private	N
TreVisitDescBranchInOrder	Visit function: descendants	Tre - 105	Private	N
TreVisitDescInOrder	Visit function: descendants	Tre - 107	Private	N
TreVisitDescInOrderBwd	Visit function: descendants	Tre - 109	Private	N
TreVisitDescPreOrder	Visit function: descendants	Tre - 111	Private	N
TreVisitInOrder	Visit function: in-order	Tre - 113	Private	N
TreVisitInOrderBwd	Visit function: in-order	Tre - 115	Private	N
TreVisitLeaves	Visit function: leaves	Tre - 117	Private	N
TreVisitParents	Visit function: nearest parents first	Tre - 119	Private	N
TreVisitPreOrder	Visit function: pre-order	Tre - 121	Private	N
TreVisitRange	Visit function: range	Tre - 123	Private	N
TreVisitSuccPreOrder	Visit function: all successors	Tre - 125	Private	N
TreVisitSuccessors	Visit function: successors	Tre - 127	Private	N

# Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
---------------	-------------	------	-------	-------

ObjSendMessage	Send message to object	Obj - 21	Public	N
ObjSendMessageReturnInt	Send message to object, return Int	Obj - 22	Public	N
ObjSendMessageReturnPtr	Send message to object, return pointer	Obj - 23	Public	N

StrBasicExtract	Extract a string	N/A	Undoc	N
StrExtract	Extract string as specified	Str - 2	Public	N
StrFromDate	Fill string with a date	Str - 4	Public	N
StrFromMediumInt	Integer to string	Str - 6	Public	N
StrInit	Init string to zero	Str - 8	Public	Y
StrReplaceSubStr	Replace sub-string in string	Str - 9	Public	N
StrSet	Copy string to another	Str - 11	Public	Y
StrSqueeze	Removes any character from string	Str - 13	Public	N
StrToDate	Parse a date string for year, month, day	Str - 15	Public	N
StrToLower	Change case of string to lower	Str - 17	Public	Y
StrToMediumInt	String to integer	Str - 19	Public	N
StrToUpper	Change case of string to upper	Str - 21	Public	Y

TreAsDll	Return node as list	Tre - 2	Private	Y
TreAsLel	Return node as list element	Tre - 3	Private	Y
TreAsObj	Return node as object	Tre - 4	Private	Y
TreClear	Clear the tree	Tre - 5	Public	N
TreClient	Return client of node	Tre - 6	Public	Y
TreClientFindChild	Visit search function: children	Tre - 7	Public	N
TreClientFirstChild	Return first child node as client	Tre - 9	Public	Y
TreClientLastChild	Return last child node as client	Tre - 11	Public	Y
TreClientLastLeaf	Return last leaf node as client	Tre - 13	Public	Y
TreClientNext	Return next node as client	Tre - 15	Public	Y
TreClientNextPreOrder	Return next PreOrder client node	Tre - 17	Public	Y
TreClientNextUncle	Return next uncle as client	Tre - 19	Public	Y
TreClientParent	Return parent node	Tre - 21	Public	Y
TreClientPrev	Return prev client	Tre - 23	Public	Y
TreClientPrevPreOrder	Return previous client PreOrderly	Tre - 25	Public	Y
TreClientVisitBranchInOrder	Visit function: branch in-order	Tre - 27	Public	N
TreClientVisitChildren	Visit function: all children	Tre - 29	Public	N
TreClientVisitChildrenBwd	Visit function: all children	Tre - 31	Public	N
TreClientVisitDescBranchInOrder	Visit function: descendants	Tre - 33	Public	N
TreClientVisitDescInOrder	Visit function: descendants	Tre - 35	Public	N
TreClientVisitDescInOrderBwd	Visit function: descendants	Tre - 37	Public	N
TreClientVisitDescLeaves	Visit function: descendants	Tre - 39	Private	N
TreClientVisitDescPreOrder	Visit function: descendants	Tre - 41	Public	N
TreClientVisitInOrder	Visit function: in-order	Tre - 43	Public	N
TreClientVisitInOrderBwd	Visit function: in-order	Tre - 45	Public	N

# Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
MmsGetMethod	Get the method	Mms - 2	Public	N
MmsGetSelector	Get the name of selector	Mms - 6	Public	N
MmsGetSuper	Get name of superclass	Mms - 4	Public	N
MmsPrint	Display the instance	Mms - 7	Public	N
MmsValidate	Validate the instance	Mms - 8	Public	N

MscGetMetaClass	Get the MetaClass	Msc - 3	Public	N
MscGetName	Get the name of superclass	Msc - 2	Public	N
MscGetOffset	Get offset of superclass	Msc - 4	Public	N
MscPrint	Display the contents of the MetaSuperClass	Msc - 5	Public	N
MscValidate	Validate that the instance is valid	Msc - 6	Public	N

MsgDeInit	Deinit instance	Msg - 2	Public	N
MsgGetMethod	Return method	Msg - 4	Public	N
MsgGetOffset	Return the object offset	Msg - 3	Public	N
MsgGetSelector	Return selector	Msg - 5	Public	N
MsgInit	Initialize instance	Msg - 6	Public	N
MsgPrint	Print the instance	Msg - 7	Public	N
MsgSend	Send message to object	Msg - 8	Public	N
MsgSendReturnInt	Send message to object, return int	Msg - 9	Public	N
MsgSendReturnPtr	Send message to object, return pointer	Msg - 10	Public	N
MsgSetSuperOffsetAndMethod	Override the method	N/A	Undoc	N

ObjDeInit	Deinitialize instance	Obj - 2	Public	N
ObjDestroy	Deinitialize and deallocate object	Obj - 3	Public	N
ObjGetClient	Return a client	Obj - 4	Public	N
ObjGetClientOrNull	Return client if non-NULL	Obj - 6	Public	N
ObjGetCls	Return class	Obj - 8	Public	N
ObjGetClsName	Return class name	Obj - 9	Public	N
ObjGetImmediateClient	Return immediate client	Obj - 10	Public	N
ObjGetMethodAndOffset	Return method and client offset	Obj - 11	Public	N
ObjGetNthSuperObject	Return nth superobject	Obj - 13	Public	N
ObjGetRootClient	Get root client	Obj - 14	Public	N
ObjGetRootClientSize	Get root object size	Obj - 15	Public	N
ObjGetSize	Get object size	Obj - 16	Public	N
ObjGetSubObjectOffset	Return offset to immediate subobject	Obj - 17	Public	N
ObjInit	Initialize object	Obj - 18	Public	N
ObjIsRoot	Return True if is root subobject	Obj - 19	Public	N
ObjRespondsToSelector	Does object understand message?	Obj - 20	Public	N

# Quick Reference Guide to Classes

---

Function Name	Description	Page	Scope	Macro
LelDestroy	Deinitialize list element object and free space	Lel - 27	Public	N
LelGetClient	Return client	Lel - 28	Public	Y
LelGetDll	Return list object	Lel - 29	Friend	Y
LelGetNext	Return next element	Lel - 30	Friend	Y
LelGetNthSuccessor	Return Nth element	Lel - 31	Friend	N
LelGetPrev	Return previous element	Lel - 32	Friend	Y
LelInit	Is element in list	Lel - 33	Public	N
LelPasteRangeAfter	Append element(s) to list	Lel - 34	Public	N
LelPasteRangeBefore	Insert element(s) to list	Lel - 36	Public	N
LelPasteRangeToList	Insert element(s) to list	N/A	Undoc	N
LelSendDestroy	Send message for list element destruction	Lel - 38	Public	N
LelTest	Test for valid list element	Lel - 39	Public	N
LelVisitRange	Visit function for range	Lel - 40	Private	N

MclCreateClass	Create a class instance	Mcl - 2	Private	N
MclDestroyClass	Destroy a class instance	Mcl - 3	Friend	N
MclFindSelector	Find a selector index given its name	Mcl - 5	Public	N
MclFindSuperClass	Find the superclass index given its name	Mcl - 6	Public	N
MclGetClassName	Get the class name	Mcl - 7	Public	N
MclGetClassSize	Return the size of a class instance	Mcl - 8	Public	N
MclGetMessageCount	Return the number of messages	Mcl - 9	Public	N
MclGetNthMms	Return the Nth MetaMessage	Mcl - 10	Public	N
MclGetNthOffset	Return the Nth SuperClass offset	Mcl - 11	Public	N
MclGetNthSuper	Return the MetaClass	Mcl - 12	Public	N
MclGetSuperClassCount	Return the number of superclasses	Mcl - 13	Public	N
MclPrint	Print the MetaClass instance	Mcl - 14	Public	N
MclSendCreateClass	Return a new instance of the MetaClass	Mcl - 15	Public	N
MclSendDestroyClass	Deallocate an instance of a Class	Mcl - 16	Public	N
MclValidate	Validate that the instance is valid	Mcl - 17	Public	N
MclValidateMessages	Validate the MetaMessages	N/A	Undoc	N
MclValidateSuperClasses	Validate the MetaSuperClasses	N/A	Undoc	N

MemClear	Clears array elements	Mem - 2	Public	N
MemCopy	Copy array elements within the array	Mem - 3	Public	N
MemCutNSet	Delete elements, shift remaining	Mem - 5	Public	N
MemDestroy	Deallocate memory	Mem - 6	Public	N
MemDuplicate	Copy array into another	Mem - 7	Public	N
MemNew	Allocate memory	Mem - 8	Public	N
MemPasteNSet	Expand array, shift remaining	Mem - 9	Public	N
MemSetChr	Set elements to character	Mem - 10	Public	N

# Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
JulCopy	Copy julian day	Jul - 14	Public	Y
JulDateStrToJulian	Date string to julian day	Jul - 16	Public	N
JulDayOfWeek	Day number in week	Jul - 18	Public	Y
JulDayOfYear	Day number in year	Jul - 20	Public	N
JulDaysInMonth	Days in month	Jul - 22	Public	N
JulDaysInQuarter	Days in quarter	Jul - 24	Public	N
JulDaysInYear	Days in year	Jul - 26	Public	N
JulDiff	Days between two dates	Jul - 28	Public	Y
JulDiffL	Days between two dates (long)	Jul - 30	Public	Y
JulGetSystemJulianDay	System date as julian day	Jul - 32	Public	N
JulInit	Set to beginning of calendar January 1, 1583	Jul - 34	Public	Y
JulIsLeapYear	Is date in leap year	Jul - 36	Public	N
JulIsMaxValue	Is date maximum julian value	Jul - 38	Public	Y
JulMax	The maximum of two julian dates	Jul - 40	Public	Y
JulMin	The minimum of two julian dates	Jul - 42	Public	Y
JulMonthDayDiff	Days between date and a day/month	Jul - 44	Public	N
JulMonthString	Fill string with month and year	Jul - 46	Public	N
JulQuarterString	Fill string with quarter and year	Jul - 48	Public	N
JulSameDayMonth	Two dates same day and month	Jul - 50	Public	N
JulSetMaxDate	Set date to maximum value	Jul - 52	Public	N
JulToCalendar	Julian day to day, month, year	Jul - 54	Public	N
JulToDateStr	Fill date string of specified format	Jul - 56	Public	N
JulValidateDate	Validate date passed as string	Jul - 58	Public	N
JulWeekString	Fill string with day and month	Jul - 60	Public	N
JulYearString	Fill string with year	Jul - 62	Public	N

LelAsObj	Return element as object	Lel - 2	Private	Y
LelClientCount	Return count for client and successors	Lel - 3	Public	N
LelClientDll	Return client of list	Lel - 5	Public	Y
LelClientFindRange	Visit search function: range	Lel - 6	Public	N
LelClientNext	Return client of next element	Lel - 8	Public	Y
LelClientPrev	Return client of previous element	Lel - 9	Public	Y
LelClientVisitBwd	Visit function: client and predecessors	Lel - 10	Public	N
LelClientVisitFwd	Visit function: client and successors	Lel - 12	Public	N
LelClientVisitPredecessors	Visit function: predecessors	Lel - 14	Public	N
LelClientVisitRange	Visit function: range	Lel - 16	Public	N
LelClientVisitSuccessors	Visit function: successors	Lel - 18	Public	N
LelCountRange	Count elements	Lel - 20	Public	N
LelCut	Cut element from list	Lel - 22	Public	N
LelCutRange	Cut element(s) from list	Lel - 24	Public	N
LelCutRangeFromList	Cut element(s) from list	N/A	Undoc	N
LelDeInit	Deinitialize list element object	Lel - 26	Public	N

# Quick Reference Guide to Classes

---

Function Name	Description	Page	Scope	Macro
EdgGetClient	Return client of edge	Edg - 19	Public	Y
EdgGetGrf	Return graph	Edg - 20	Friend	Y
EdgGetInVtx	Return incoming vertex	Edg - 21	Friend	Y
EdgGetNextIn	Return next incoming edge	Edg - 23	Friend	Y
EdgGetNextOut	Return next outgoing edge	Edg - 25	Friend	Y
EdgGetOutVtx	Return outgoing vertex	Edg - 27	Friend	Y
EdgGetVertices	Return vertices to edge	Edg - 29	Public	Y
EdgHasVertices	Does edge have any vertices	Edg - 31	Public	N
EdgInGrf	Is edge in graph	Edg - 32	Public	Y
EdgInit	Initialize the edge object	Edg - 33	Public	N
EdgSendDestroy	Send message for edge destruction	Edg - 34	Public	Y
EdgUpdateInVtx	Replace incoming vertex	Edg - 35	Public	N
EdgUpdateOutVtx	Replace outgoing vertex	Edg - 37	Public	N

GrfAnyCycles	Check graph for cycles	Grf - 2	Public	Y
GrfAsEdgDll	Return List of edges	Grf - 4	Friend	N
GrfAsObj	Return graph as object	Grf - 5	Private	N
GrfAsVtxDll	Return List of vertices	Grf - 6	Friend	N
GrfBasicTopologicalSort	Do topological sort of graph	N/A	Undoc	N
GrfClear	Clear the graph	Grf - 7	Public	N
GrfCountEdg	Count edges of graph	Grf - 8	Public	Y
GrfCountVtx	Count vertices of graph	Grf - 10	Public	Y
GrfDeInit	Deinitialize Graph object	Grf - 12	Public	N
GrfDestroy	Deinitialize Graph object and free space	Grf - 13	Public	N
GrfDoTopologicalSort	Do topological sort of graph	Grf - 14	Public	Y
GrfFindEdgClient	Visit search function: edges	Grf - 16	Public	N
GrfFindVtxClient	Visit search function: vertices	Grf - 18	Public	N
GrfGetClient	Return client of graph	Grf - 20	Public	N
GrfInit	Initialize Graph object	Grf - 21	Public	N
GrfSendDestroy	Send message for graph destruction	Grf - 22	Public	N
GrfVisitEdgClient	Visit function: edges	Grf - 23	Public	N
GrfVisitVtxClient	Visit function: vertices	Grf - 25	Public	N
GrfVisitVtxClientInTopOrderBwd	Visit function: backward topological order	Grf - 27	Public	N
GrfVisitVtxClientInTopOrderFwd	Visit function: forward topological order	Grf - 29	Public	N

JulAddDays	Add/subtract days to date	Jul - 2	Public	Y
JulAddDaysL	Add/subtract days to date (long)	Jul - 4	Public	Y
JulAddMonths	Add/subtract months to date	Jul - 6	Public	N
JulAddQuarters	Add/subtract quarters to date	Jul - 8	Public	N
JulAddYears	Add/subtract years to date	Jul - 10	Public	N
JulCalendarToJulian	Day, month, year to julian day	Jul - 12	Public	N

## Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
DpaAppend	Append an element	Dpa - 2	Public	N
DpaClear	Clear dynamic array	Dpa - 4	Public	N
DpaCount	Visit function: count True returns	Dpa - 5	Public	N
DpaCountRange	Visit function: range with return checking	Dpa - 7	Public	N
DpaDeInit	Deinitialize the dynamic array object	Dpa - 9	Public	N
DpaDelete	Delete element(s)	Dpa - 10	Public	N
DpaDestroy	Deinitialize array object and free space	Dpa - 12	Public	N
DpaExpand	Paste Null element(s) into array	Dpa - 13	Public	N
DpaFind	Find index returning True	Dpa - 15	Public	N
DpaFindPtrBwd	Find index with matching pointer	Dpa - 17	Public	N
DpaFindPtrFwd	Find index with matching pointer	Dpa - 19	Public	N
DpaFindRangeBwd	Find index returning True for range	Dpa - 21	Public	N
DpaFindRangeFwd	Find index returning True for range	Dpa - 23	Public	N
DpaGetLast	Return last element in array	Dpa - 27	Public	N
DpaGetNth	Return Nth array element	Dpa - 25	Public	N
DpaGetSize	Return number of elements	Dpa - 29	Public	N
DpaInit	Initialize the edge object	Dpa - 31	Public	N
DpaNewArray	Create a new array	N/A	Undoc	N
DpaResize	Resize the array	N/A	Undoc	N
DpaSetNth	Set Nth element of array	Dpa - 33	Public	N
DpaSetRegionNull	Make region of elements Null	Dpa - 35	Public	N
DpaSetSize	Set array size to N elements	Dpa - 37	Public	N
DpaShiftDown	Shift down N elements in array	Dpa - 38	Public	N
DpaShiftUp	Shift up N elements in array	Dpa - 40	Public	N
DpaVisit	Visit function: all elements	Dpa - 42	Public	N
DpaVisitClient	Visit function: all elements	Dpa - 44	Public	N
DpaVisitRange	Visit function: range of elements	Dpa - 46	Public	N
DpaVisitRegion	Visit function: region of elements	Dpa - 48	Public	N
DpaVisitSelfAndSuccessors	Visit function: client and successors	Dpa - 50	Public	N

EdgAsGrfLel	Return graph list element for edge	Edg - 2	Friend	Y
EdgAsInLel	Return incoming edge list element	Edg - 3	Friend	Y
EdgAsObj	Return edge as object	Edg - 4	Private	N
EdgAsOutLel	Return outgoing edge list element	Edg - 5	Friend	Y
EdgClear	Clear the edge	Edg - 6	Public	N
EdgCompareInVtx	Compare incoming vertex	Edg - 7	Public	N
EdgConnectToGrf	Connect edge to graph	Edg - 9	Public	Y
EdgConnectToVertices	Connect edge to vertices	Edg - 11	Public	N
EdgDeInit	Deinitialize the edge object	Edg - 13	Public	N
EdgDestroy	Deinitialize edge object and free space	Edg - 14	Public	N
EdgDisconnectFromGrf	Disconnect edge from graph	Edg - 15	Public	Y
EdgDisconnectFromVertices	Disconnect edge from vertices	Edg - 17	Public	N



# Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
ClsGetSuperClassCount	Return number of superclasses	Cls - 28	Public	N
ClsGetSuperClassIndex	Return superclass index	Cls - 29	Public	N
ClsInit	Initialize the instance	Cls - 30	Public	N
ClsIsRoot	Is instance the root subclass?	Cls - 31	Public	N
ClsOffsetSupers	Offset superclasses from subclass.	N/A	Undoc	N
ClsPrint	Print the instance	Cls - 32	Public	N
ClsSendDestroy	Send destroy message to instance	Cls - 33	Public	N
ClsSendMessage	Send object a message	Cls - 34	Public	N
ClsSendMessageReturnInt	Send object message, returns MediumInt	Cls - 36	Public	N
ClsSendMessageReturnPtr	Send object message, returns Void *	Cls - 38	Public	N
ClsSuperClassOf	Initialize sub/super relation	N/A	Undoc	N

DIIAppend	Append element to list	DII - 2	Public	N
DIIAppendLast	Make element last	DII - 4	Public	Y
DIIAsObj	Return list as object	DII - 6	Private	Y
DIIClear	Clear the list	DII - 7	Public	N
DIICut	Cut one element from list	DII - 8	Public	N
DIICutChildren	Cut all elements from list	DII - 10	Public	N
DIICutRange	Cut element(s) from list	DII - 11	Public	N
DIIDeInit	Deinitialize list object	DII - 13	Public	N
DIIDestroy	Deinitialize list object and free space	DII - 14	Public	N
DIIGetClient	Return client of list	DII - 15	Public	Y
DIIGetFirst	Return first element	DII - 16	Private	Y
DIIGetLast	Return last element	DII - 17	Private	Y
DIIGetNth	Return Nth element	DII - 18	Private	N
DIIGetSize	Get size of list	DII - 19	Public	N
DIIInit	Initialize list object	DII - 20	Public	N
DIIInsert	Insert element in list	DII - 21	Public	N
DIIInsertFirst	Make element first	DII - 23	Public	Y
DIIIsEmpty	Return True if list empty	DII - 25	Public	Y
DIILeiClientCount	Visit function: count elements conditionally	DII - 26	Public	N
DIILeiClientFind	Visit search function: all elements	DII - 28	Public	N
DIILeiClientFirst	Return client of first element	DII - 30	Public	Y
DIILeiClientGetNth	Return Nth client	DII - 31	Public	N
DIILeiClientLast	Return client of last element	DII - 33	Public	Y
DIILeiClientVisitBwd	Visit function: all elements	DII - 34	Public	N
DIILeiClientVisitFwd	Visit function: all elements	DII - 36	Public	N
DIINotifyCutRange	Cut elements	DII - 38	Friend	N
DIINotifyPasteRange	Paste elements	DII - 39	Friend	N
DIIPasteRangeAfter	Paste element(s) to list	DII - 40	Public	N
DIIPasteRangeBefore	Paste element(s) in list	DII - 42	Public	N
DIIPasteRangeFirst	Paste element(s) to be first in list	DII - 44	Public	Y
DIIPasteRangeLast	Paste element(s) to end of list	DII - 46	Public	Y
DIISendDestroy	Send message for list destruction	DII - 48	Public	N

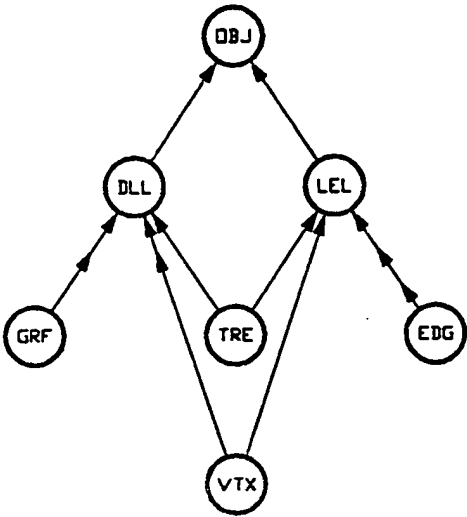
# Quick Reference Guide to Classes

Function Name	Description	Page	Scope	Macro
BlkClear	Clear instance	Blk - 2	Public	N
BlkDeInit	Deinitialize instance	Blk - 3	Public	N
BlkExecute	Execute method with parameters	Blk - 4	Public	N
BlkExecuteRetBool	Execute the method, return int	Blk - 5	Public	N
BlkExecuteRetDataPtr	Execute method, return mem pointer	Blk - 6	Public	N
BlkExecuteRetFuncPtr	Execute method, return function ptr	Blk - 7	Public	N
BlkExecuteRetInt	Execute method, return int	Blk - 8	Public	N
BlkHasMethod	Return True if instance has non-NULL method	Blk - 9	Public	N
BlkInit	Initialize instance	Blk - 10	Public	N
BlkPrint	Print contents of instance	Blk - 11	Public	N
BlkPushDataPtr	Save pointer parameter	Blk - 12	Public	N
BlkPushFuncPtr	Save function pointer parameter.	Blk - 13	Public	N
BlkPushLargeInt	Save LargeInt parameter	Blk - 14	Public	N
BlkPushMediumInt	Save MediumInt parameter	Blk - 15	Public	N
BlkSetMethod	Set the function to call	Blk - 16	Public	N

ClsCreateMessages	Create the Msg instances	N/A	Undoc	N
ClsCreateObject	Create a new Object instance	Cls - 2	Public	N
ClsCreateSupers	Create the superclass instances	N/A	Undoc	N
ClsDeInit	Deinitialize the instance	Cls - 3	Public	N
ClsDestroy	Deallocate the instance	Cls - 4	Public	N
ClsDestroyMessages	Destroy the messages	N/A	Undoc	N
ClsDestroyObject	Deallocate the object instance	Cls - 5	Public	N
ClsDestroySuperClasses	Destroy the superclasses	N/A	Undoc	N
ClsFindMsg	Find message given selector name	Cls - 7	Public	N
ClsFindSelectorIndex	Find index of selector given name	Cls - 8	Public	N
ClsFindSuperClass	Find superclass given name	Cls - 10	Public	N
ClsGetMessageCount	Get number of messages	Cls - 11	Public	N
ClsGetMethodAndOffset	Return method and sub/super offset	Cls - 12	Public	N
ClsGetName	Return name of the instance	Cls - 14	Public	N
ClsGetNthMsg	Get pointer to nth Message	Cls - 15	Public	N
ClsGetNthSuperClass	Get pointer to Nth superclass	Cls - 16	Public	N
ClsGetObjectSize	Return size of an object instance	Cls - 17	Public	N
ClsGetOffsetForMsg	Return sub/super offset of object	Cls - 18	Public	N
ClsGetOffsetOfNthSuper	Return offset of nth superobject	Cls - 20	Public	N
ClsGetRootSubClass	Get the root subclass	Cls - 22	Public	N
ClsGetRootSubObjectOffset	Return offset of root subobject	Cls - 23	Public	N
ClsGetRootSubObjectSize	Return size of the root subobject	Cls - 24	Public	N
ClsGetSize	Return size of the instance	Cls - 25	Public	N
ClsGetSubClass	Return subclass	Cls - 26	Public	N
ClsGetSubObjectOffset	Return offset of a subobject	Cls - 27	Public	N

# Class Inheritance Diagram

## C+0 Class Library

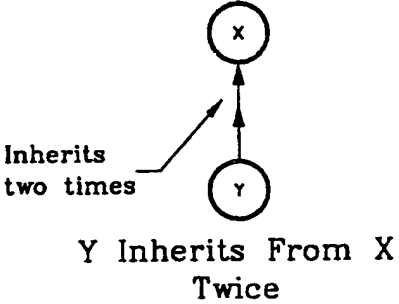
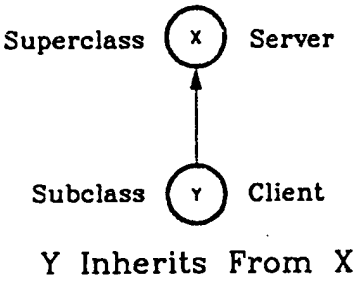


## Class Inheritance Diagram



## Primitive Classes

### LEGEND



## Introduction

The C+O Reference Manual provides a concise description of each of the functions provided in the class library. The User's Guide contains installation, tutorial, descriptions of each class, and error messages. It is recommended that you begin by reading the User's Guide, then use this manual to learn the specifics of a particular function.



# ***Table of Contents***

## **REFERENCE MANUAL**

<b>Introduction</b> . . . . .	<b>i</b>
<b>Class Inheritance Diagram</b> . . . . .	<b>ii</b>
<b>Quick Reference: Class Functions</b> . . . . .	<b>iii - xiv</b>
<b>C+O Datatypes Table</b> . . . . .	<b>xv - xviii</b>

## **Class Reference**

Blk - Block . . . . .	1-16
Cls - Class . . . . .	1-40
Dll - List . . . . .	1-48
Dpa - DynamicArray . . . . .	1-52
Edg - Edge . . . . .	1-38
Grf - Graph . . . . .	1-30
Jul - JulianTime . . . . .	1-64
Lel - ListElement . . . . .	1-42
Mcl - MetaClass . . . . .	1-18
Mem - Memory . . . . .	1-10
Mms - MetaMessage . . . . .	1-8
Msc - MetaSuperClass . . . . .	1-6
Msg - Message . . . . .	1-10
Obj - Object . . . . .	1-24
Str - String . . . . .	1-22
Tre - Tree . . . . .	1-128
Tsk - Task . . . . .	1-42
Vtx - Vertex . . . . .	1-40

Information in this document is subject to change without notice and does not represent a commitment on the part of Objective Systems. The software described in this document is furnished under a license agreement. The software may be used or copied only in accordance with the terms of the agreement. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose other than the purchaser's personal use without the written permission of Objective Systems.

**Objective Systems**  
2443 Fillmore Street  
Suite 249  
San Francisco, California 94115

Tel: 415/ 929-0964  
Fax: 415/ 929-8015

© Copyright Objective Systems 1988.  
All rights reserved

C+O™ is the trademark of Objective Systems.

This manual was produced in its entirety with the Objective Systems Object-Oriented CASE documentation tool and Microsoft Word. Output was to a PostScript printer.

**C+O Class Library  
Reference Manual  
Volume 1: Foundation Data Structures**

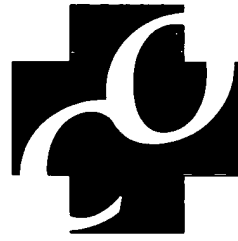
Revision 1.2 July, 1988

**For the OS/2 Operating System**

Objective Systems







**C+O CLASS LIBRARY  
FOUNDATION  
DATA STRUCTURES**

*OS/2 Version/ Volume One  
Reference Manual*

**EXHIBIT F**

Title: Business Analysis & Management Systems  
Utilizing Enterprise Metrics

Inventors: Michael M. Mann & Arne Haugland

Attys.: Fulwider Patton et al. Dkt. # 65568/ENCMP